

LA PROGRAMMATION DES CARTES IEEE POUR PC

MODULE LOGICIEL I2C EN LANGAGE C

APPLICATIONS DU SSM 2024, QUADRUPLE VCA

UN LECTEUR-PROGRAMMATEUR DE CARTES I2C

LES PÉRIPHÉRIQUES PROGRAMMABLES WSI

RÉALISATION DE LA CARTE P.I.P, DIGIT 2000

JUILLET 1992

BELGIQUE : 155 FB - LUXEMBOURG : 155 FL - SUISSE : 6,30 FS - ESPAGNE : 450 Ptas - CANADA : $ 4.25

LE PCF 8584, INTERFACE BUS PARALLELE-I2C



NUMERO 536 - JUILLET 92 - ISSN 1144-5742

électronique RADIO PLANS

ELECTRONIQUE APPLICATIONS

MENSUEL édité par la Société Parisienne d'Édition
Société anonyme au capital de 1 950 000 F
Siège social
Direction-Rédaction-Administration-Ventes :
2 à 12, rue de Bellevue, 75940 Paris Cedex 19
Tél. : 42.00.33.05
Télex : PGV 220409 F - Télécopie : 42.41.89.40
Président-Directeur Général,
Directeur de la Publication :
J.-P. VENTILLARD
Directeur de la Rédaction :
Bernard FIGHIERA
Rédacteur en chef :
Claude DUCROS
Publicité : Société Auxiliaire de Publicité
70, rue de Compans, 75019 Paris
Tél. : 42.00.33.05 - C.C.P. 37-93-60 Paris
Directeur commercial : J.-P. REITER
Chef de publicité : Francine FIGHIERA
Assistée de : Laurence BRESNU et de Murielle KAISER
Marketing : Jean-Louis PARBOT
Directeur des ventes : Joël PETAUTON
Inspecteur des ventes : Société PROMEVENTE
M. Michel IATCA
24-26, bd Poissonnière, 75009 Paris.
Tél. : 45.23.25.60 - Fax. 42.46.98.11
Service des abonnements :
2 à 12, rue de Bellevue, 75019 Paris.
Voir notre tarif
« spécial abonnement ».
Pour tout changement d'adresse, envoyer la dernière bande accompagnée de 2,50 F en timbres.
IMPORTANT: ne pas mentionner notre numéro de compte pour les paiements par chèque postal.

Ce numéro a été tiré
à 45 700 exemplaires
Dépot légal juillet 92 - Éditeur 1690 -
Mensuel paraissant en fin de mois.
Distribué par S.A.E.M. Transport-Presse.
Photocomposition COMPOGRAPHIA - 75011 PARIS -
Imprimerie SIEP Bois-le-Roi et REG Lagny.
Photo de couverture : E. Malemanche.







SOMMAIRE

ETUDE ET CONCEPTION

82 Ensemble d'incrustation P.I.P. avec le kit DIGIT 2000 (2)

MONTAGES

40 Interface bidirectionnelle pour port parallèle PC

48 Lecteur-programmateur de carte à puce I2C

CIRCUITS D'APPLICATIONS

19 Le SSM 2024, quadruple VCA AD

25 Le PCD 8584, circuit d'interface bus parallèle 8 bits - Bus I2C

90 Des récepteurs performants avec le MC 3356 Motorola

MESURE ET INSTRUMENTATION

53 La programmation des cartes IEEE pour PC

69 Programmation du HP 34401

TECHNIQUE

33 Les mélangeurs, critères de sélection et d'application

46 ISI, logiciel de calculs techniques

COMPOSANTS ET TECHNOLOGIE

11 Les périphériques programmables de WSI

COMMUNICATION

75 Un module logiciel en C pour le bus I2C

INFOS

79 DAQ Designer : logiciel NI d'aide à la sélection d'éléments d'acquisition

80 L'AD 1671, CAN 12 bits faible coût, AD

Le 2212 oscilloscope mixte d'entrée de gamme TEKTRONIX

96 Ciné Pile : la station d'énergie 12 V de mille et une piles

97 Carte réseau FIP, DIGIMETRIE

A propos de la gestion I2C centralisée pour caravanes

Le LT 1116, comparateur 5 V ultra-rapide

Ont participé à ce numéro :
J. Alary, C. Basso, J.-Y. Bedu, B. Delabre, F. et G. de Dieuleveult, X. Fenard, A. Garrigou, P. Gueulle, C. Lefèbvre, D. Paret.

Kits PC/AT 6809/68000

Des écrans superbes, une convivialité étonnante...

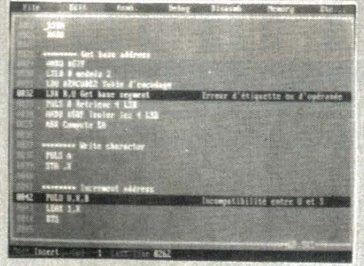
Pourquoi de nombreuses académies choisissent les kits DATA RD plutôt que ceux de la concurrence pour équiper les lycées techniques ? Les raisons sont simples : les kits DATA RD sont extrêmement pédagogiques, très faciles à utiliser, performants et très compétitifs.

Gamme K32

Ce sont des kits autonomes, comprenant éditeur, assembleur 2 passes, débogueur... La version industrielle est dotée de relais, Darlingtons, CDA/CAD, opto-coupleurs, bread-board... et les TP sont très rapides à préparer. Ces kits dialoguent tous avec un PC.

Gamme PC9-P68K

Ces cartes sont vendues avec un logiciel PC du genre "turbo", très facile à utiliser : menus déroulants, écrans multiples, aide en ligne, disque dur... La nouvelle version de cet environnement intégré PC/AT, dont le source ne fait pas moins de 47000 lignes, comprend une gestion de disques, un éditeur, un macro-assembleur, un débogueur, un désassembleur... Examinez les photos d'écran : elles sont superbes et très pédagogiques, comme le reste du logiciel. Vous ne trouverez pas chez la concurrence un environnement aussi convivial.





Assemblez et corrigez facilement

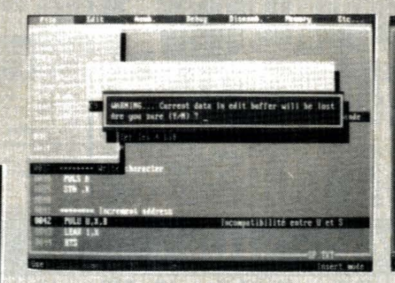
Nos assembleurs sont très performants : macros, assemblage conditionnel, étiquettes locales, indication du nombre de cycles etc... Quant aux listings sur écran ou imprimante, ils sont superbes. De plus, en cas d'erreurs, lorsque vous revenez dans l'éditeur, le curseur se place automatiquement sur les lignes erronées. Les erreurs sont mises en rouge et commentées en français dans l'éditeur. Enfin, notez que le package éditeur-assembleur-désassembleur est également vendu séparément pour un prix très compétitif.

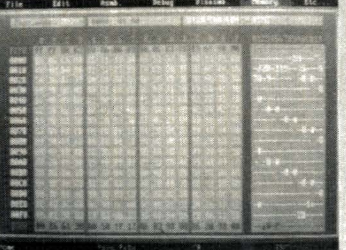
Un débogage très simple

En fait, vous n'avez rien à faire : les registres, la RAM, le contenu de la pile, les interfaces... sont visualisés en permanence. De plus, les registres modifiés à la fin d'une instruction apparaissent en surbrillance. Et la lecture-écriture mémoire est aussi simple : pointez et entrez la nouvelle valeur, c'est tout... La partie débogage à elle seule justifie l'achat de kits DATA RD : c'est une merveille de convivialité.

Alors comparez...

Vous pensez que tout ça, c'est de la "pub"... Alors sachez que nous vous prêtons (*) gracieusement et sans obligation d'achat un kit pour 15 jours, juste le temps de l'évaluer ou de le comparer avec la concurrence (ce que nous vous conseillons très vivement). Vous vous apercevrez alors que cette pub est bien le reflet de la réalité.



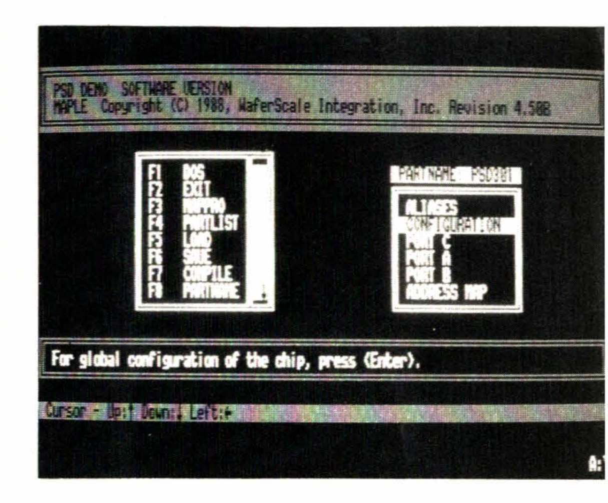


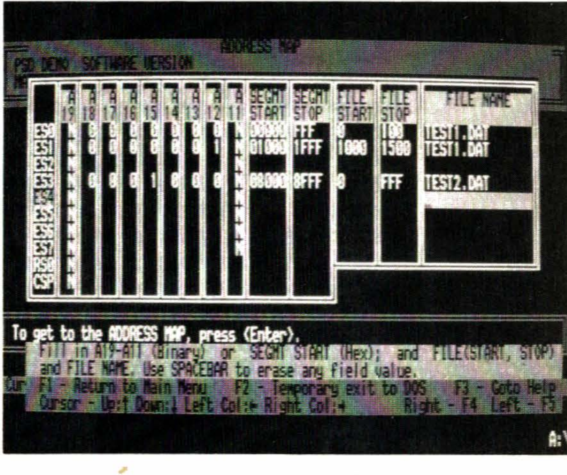
DATA R.D.
14, rue Gaspard Monge
Z.A. de l'Armailler
26500, BOURG-Lès-VALENCE
(France) *Tél. 75-83-27-25*

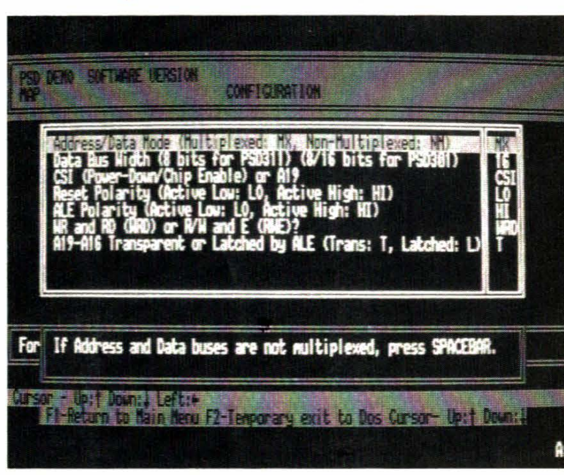
(*) Selon disponibilité, sur demande écrite du Chef des Travaux ou assimilé.
Publicité non contractuelle.
TURBO est une marque générique de BORLAND.
Certaines innovations ont été brevetées par DATA RD.

Les périphériques programmables PSD3XX de WSI

Outre l'unité centrale, les cartes à microprocesseur rassemblent traditionnellement de multiples circuits périphériques et des mémoires.
Les microcontrôleurs intègrent en principe tout cela dans un seul et même boîtier, mais leurs ressources peuvent se révéler insuffisantes pour bien des projets d'une certaine ampleur : il faut alors leur ajouter de la mémoire ou même des périphériques, ce qui ramène au cas précédent !
L'innovation majeure introduite par WSI avec sa famille de "périphériques programmables" est de réunir dans un unique boîtier, des périphériques d'entrée-sortie et une généreuse quantité de mémoire RAM et EPROM.







La logique interne de configuration et d'adressage étant programmable tout comme un simple PLD, quelques références suffisent pour assurer la compatibilité avec pratiquement n'importe quel microprocesseur ou microcontrôleur.
Des solutions à deux boîtiers seulement sont donc envisageables, même pour des applications d'un haut degré de complexité.

TOUT SAUF L'UNITE CENTRALE

L'architecture typique d'un système microprogrammé est bien connue : une unité centrale, de la mémoire ROM et RAM, et différents périphériques. Le tout relié par des bus de données, d'adresse et de contrôle munis de circuits de décodage ou de sélection.
Si l'unité centrale est un simple microprocesseur, il est habituel de lui associer au moins deux boîtiers de mémoire, un ou deux boîtiers d'entrée-sortie, et soit un ou deux PAL de décodage, soit cinq ou six composants CMOS ou TTL.
L'utilisation de microcontrôleurs permet de concentrer pratiquement tout cela dans un unique boîtier, du moins tant que l'on n'a pas besoin de trop de capacité mémoire ou d'un trop grand nombre de lignes d'entrée-sortie.
Si l'on doit ajouter de la mémoire externe, il faut la relier au microcontrôleur par des bus qui monopolisent alors une bonne partie des lignes d'entrée-sortie disponibles.
Leur éventuelle reconstitution nécessite alors plusieurs boîtiers, auxquels il faut ajouter le démultiplexeur habituellement utilisé pour interfacer la mémoire (voir l'exemple de la **figure 1**).
A vrai dire, la solution strictement "monochip" ne concerne qu'une minorité d'applications relativement simples.
Or, il devient de plus en plus souvent nécessaire de développer des systèmes complexes mais miniaturisés et à faible consommation, et ce dans des délais toujours plus courts : des exemples typiques sont les radiotéléphones cellulaires, les contrôleurs de disques durs, les modems, l'instrumentation, les périphériques d'ordinateurs, et d'une façon générale beaucoup

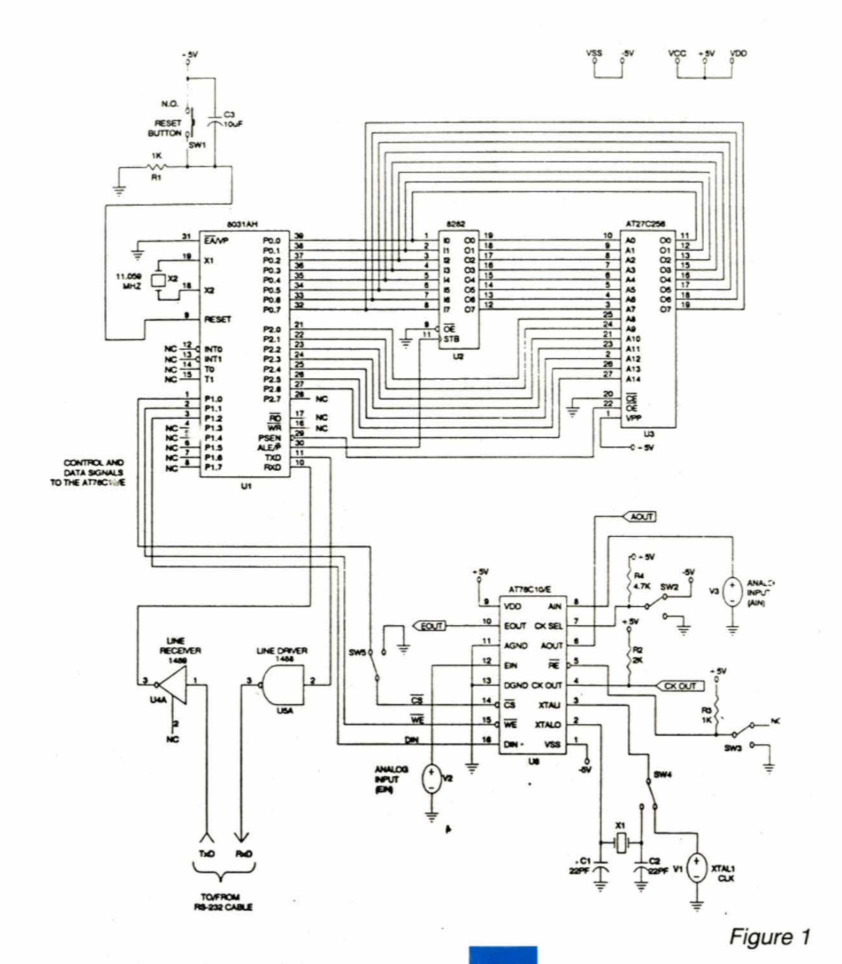


Figure 1

de systèmes industriels, automobiles, médicaux ou militaires.
Les périphériques programmables de WSI (Wafer Scale Integration) permettent de concilier tous ces impératifs, et représentent donc une innovation importante pour le développeur de systèmes de moyenne ou grande complexité.
Les composants de la famille PSD 3XX rassemblent en effet dans un seul boîtier (céramique à fenêtre ou OTP) tout ce qu'il faut pour ajouter des ports d'entrée-sortie, 256 à 1024 k-bits d'EPROM, et 16 k-bits de RAM statique à un microprocesseur ou microcontrôleur pratiquement quelconque : 6805, 68HC11, 68000/10/20, 8031/8051, 8096/8098, 80186/88, 80196/98, Z80, Z8, etc. ou même à un DSP.
Certaines références offrent par ailleurs une possibilité d'extension de l'espace adressable d'origine d'un microcontrôleur par une technique de pagination : jusqu'à 16 pages d'un mégabit !
Mais comment six modèles seulement peuvent-ils offrir une telle souplesse ?

Tout simplement parce que la logique interne des PSD 3XX est bâtie autour de réseaux logiques programmables comparables à des PAL : un logiciel spécial permet à l'utilisateur de décrire son système à l'aide du premier PC venu, puis un programmateur approprié personnalise le composant en conséquence.
Un atout majeur de cette démarche est que les PSD 3XX disposent d'un "bit de sécurité" comme n'importe quel PAL : une fois programmé, il interdit toute relecture des données de configuration et donc toute tentative de "reverse engineering".
Même le contenu de l'EPROM, pourtant accessible librement par l'unité centrale, peut être considéré comme protégé : en effet, comme la carte de la mémoire et des entrées-sorties ne peut être consultée, il n'est pratiquement pas possible de savoir à quoi correspondent les octets que l'on pourrait arriver à lire par un moyen ou par un autre...

Le PSD 301

La famille PSD 3XX se compose actuellement de six références, qui ne diffèrent pas très profondément les unes des autres : elles sont en fait chacune plus ou moins optimisées pour certaines catégories de processeurs, et incorporent éventuellement des perfectionnements supplémentaires.

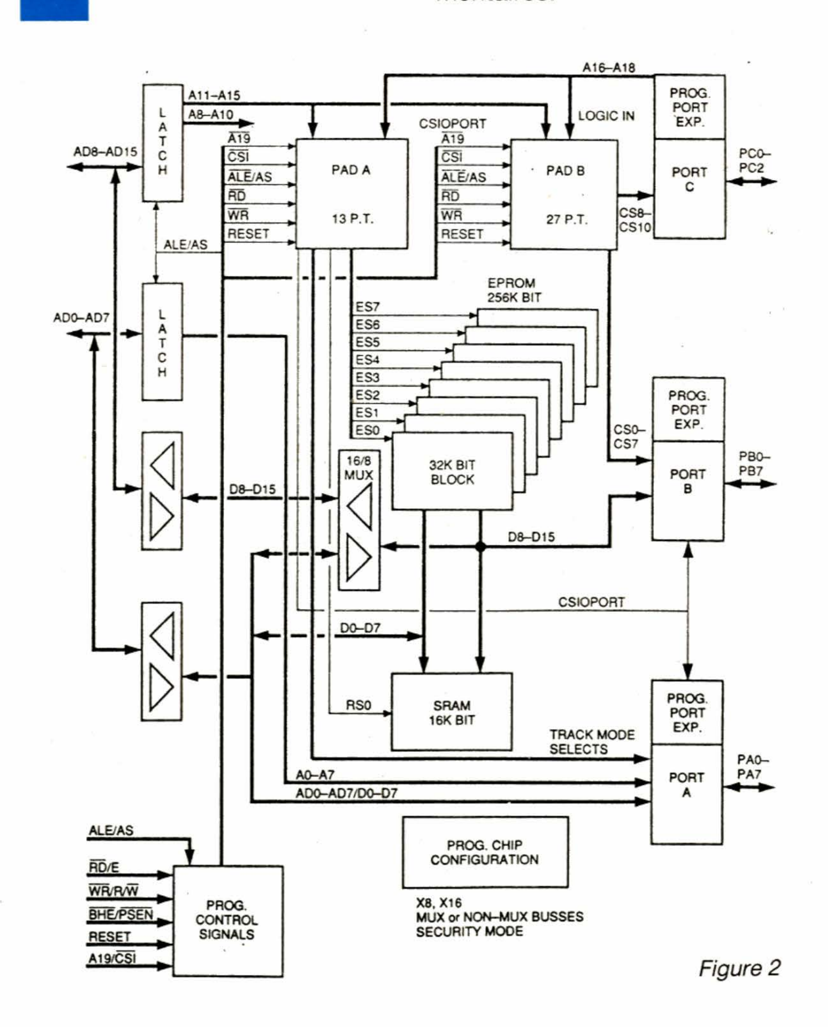


Figure 2

Le PSD 301 est le premier en date de ces composants mais reste assez représentatif de l'ensemble de la gamme, qui comprend les références suivantes : PSD 301, PSD 311, PSD 302, PSD 312, PSD 303 et PSD 313.

Le PSD 301, dont la **figure 2** reproduit le schéma synoptique, offre 19 lignes d'entrée-sortie configurables individuellement, et pouvant donc être utilisées indifféremment comme :

- extension des entrées-sorties du microcontrôleur hôte,
- entrées-sorties du décodeur d'adresses programmable,
- sorties d'adresses verrouillées,
- lignes à drain ouvert ou CMOS.

Deux réseaux logiques programmables dits "Programmable Address Decoders" (PAD A et PAD B) offrent un total de 40 termes de produits et jusqu'à 16 entrées et 24 sorties. Ils remplacent la totalité de la logique d'appoint habituelle, et offrent une capacité de décodage d'adresses d'un mégabit.

Des verrous sont prévus pour la gestion de bus de données et d'adresses multiplexés, mais le circuit reste évidemment compatible avec les bus non multiplexés.

Dans les deux cas, les bus à 8 ou 16 bits sont supportés.

La polarité des signaux ALE et RESET est librement programmable, tandis que le bus de contrôle peut être du type RD barre/WR barre ou R/W barre/E.

Une broche BHE est prévue pour le mode 16 bits, ainsi qu'une broche PSEN pour les utilisateurs de 8051.

256 k-bits d'EPROM effaçable aux UV (sauf boîtiers OTP) sont disponibles, sous la forme 32 k x 8 ou 16 k x 16.

Le tout est divisé en huit blocs pouvant être répartis librement dans l'espace adressable : cela accroît la souplesse du système et contribue à la protection du contenu de l'EPROM contre les tentatives de lecture ou de copie.

S'y ajoutent 16 k-bits de RAM statique 120 ns, organisée en 2 k x 8 ou 1 k x 16.

Enfin, des facilités sont prévues pour le partage de ressources entre plusieurs microcontrôleurs ou avec un processeur hôte.

Le cœur du PSD 301 est évidemment son réseau logique programmable, dont la **figure 3** dévoile l'organisation. On retrouve l'architecture classique des PAL (réseau programmable de portes ET suivi d'un réseau fixe de portes OU).

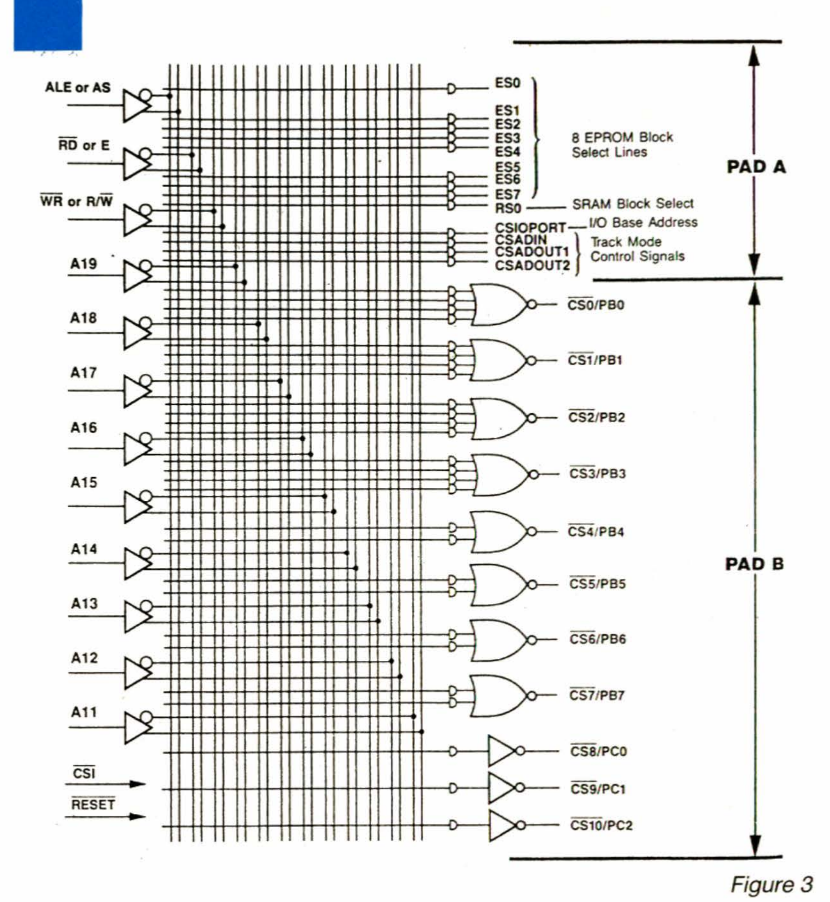
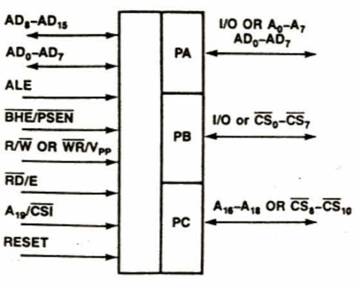
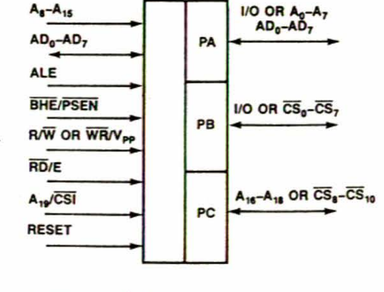


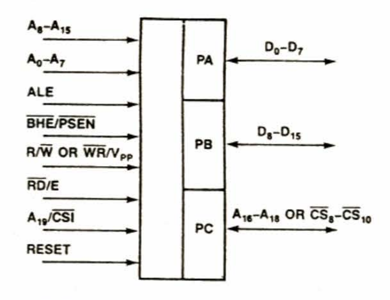
Figure 3



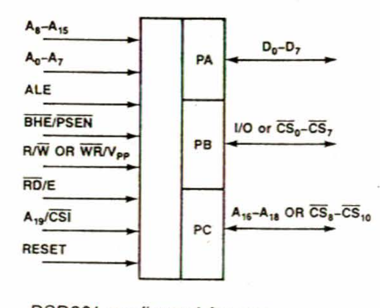
PSD301 configured for multiplexed 16-bit address/data bus



PSD301 configured for multiplexed 8-bit address/data bus.



PSD301 configured for non-multiplexed 16-bit address/data bus.



PSD301 configured for non-multiplexed 8-bit address/data bus.

Figure 4

Par rapport à un PAL courant comme le 16L8, par exemple, la matrice programmable est plus petite, mais davantage optimisée : il en résulte un nombre d'entrées et de sorties supérieur, réparti d'ailleurs en deux groupes distincts (PAD A et PAD B) dont chacun remplace pratiquement l'équivalent d'un PAL classiquement sous-employé.

C'est par simple programmation de ce réseau logique que l'on peut adapter très exactement l'architecture du PSD 301 à celle du système final, et notamment aux caractéristiques de ses bus comme le montre la **figure 4** : 8 ou 16 bits, multiplexés ou non.

Cela suppose une structure assez complexe des ports d'entrée-sortie, directement placés

sous le contrôle du réseau programmable : la **figure 5** décrit l'agencement du port A, la **figure 6** celui du port B, et la **figure 7** celui du port C.

Quarante quatre broches sont nécessaires pour assurer la liaison du PSD 301 avec le "monde extérieur", processeur compris.

Plusieurs boîtiers sont disponibles, avec ou sans fenêtre pour l'effacement aux UV : un LCC céramique ou plastique à 44 broches (**figures 8** et **9**), un QFP plastique à 52 broches (**figure 10**) et un PGA céramique à 44 broches (**figure 11**).

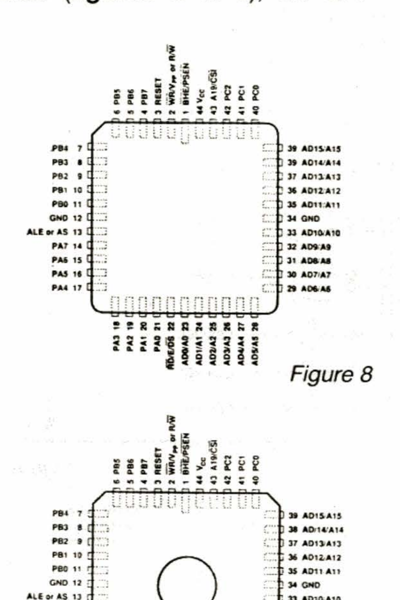


Figure 8

Figure 9

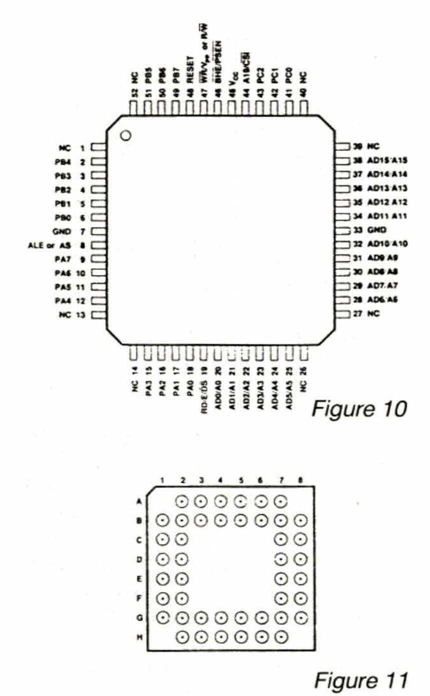


Figure 10

Figure 11

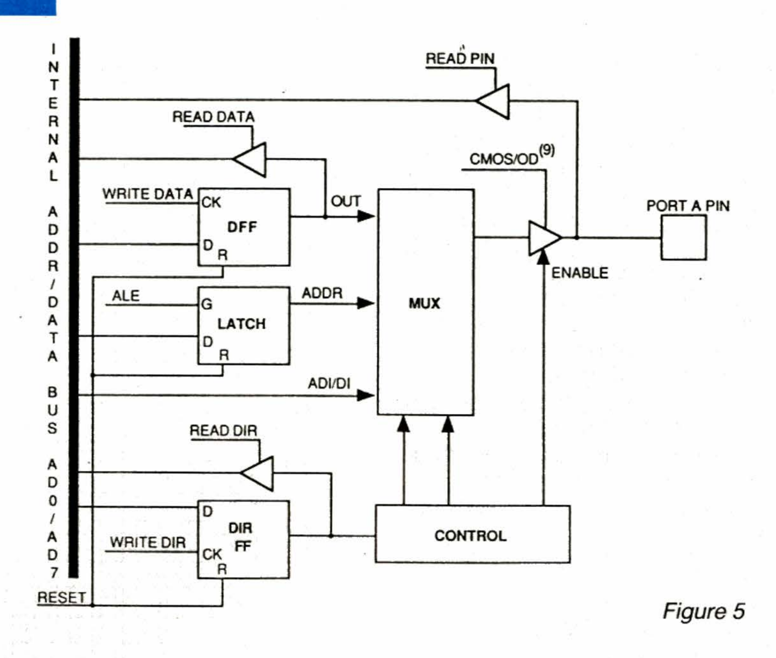


Figure 5

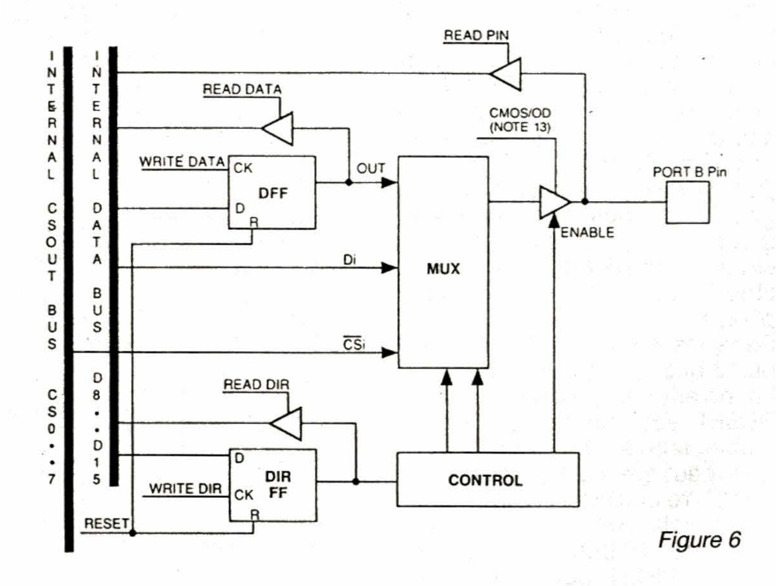


Figure 6

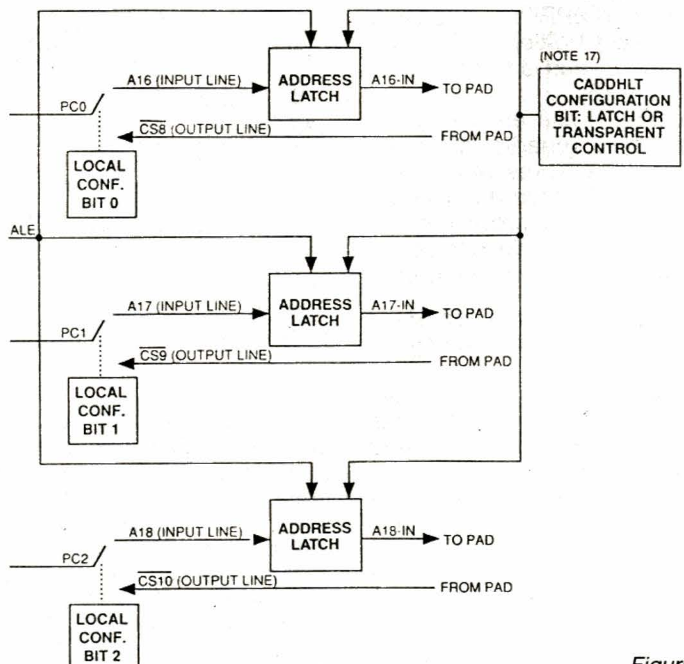


Figure 7

Le tableau de la **figure 12** résume les brochages de ces différentes exécutions, qui sont d'ailleurs compatibles avec ceux des autres références PSD 3XX.

Name	44-Pin PLDCC/ CLDCC Package	44-Pin CPGA Package	52-Pin PQFP Package
$\overline{BHE}$/$\overline{PSEN}$	1	A_5	46
$\overline{WR}$/V_{PP} or R/$\overline{W}$	2	A_4	47
RESET	3	B_4	48
PB7	4	A_3	49
PB6	5	B_3	50
PB5	6	A_2	51
PB4	7	B_2	2
PB3	8	B_1	3
PB2	9	C_2	4
PB1	10	C_1	5
PB0	11	D_2	6
GND	12	D_1	7
ALE or AS	13	E_1	8
PA7	14	E_2	9
PA6	15	F_1	10
PA5	16	F_2	11
PA4	17	G_1	12
PA3	18	G_2	15
PA2	19	H_2	16
PA1	20	G_3	17
PA0	21	H_3	18
$\overline{RD}$/E	22	G_4	19
AD0/A0	23	H_4	20
AD1/A1	24	H_5	21
AD2/A2	25	G_5	22
AD3/A3	26	H_6	23
AD4/A4	27	G_6	24
AD5/A5	28	H_7	25
AD6/A6	29	G_7	28
AD7/A7	30	G_8	29
AD8/A8	31	F_7	30
AD9/A9	32	F_8	31
AD10/A10	33	E_7	32
GND	34	E_8	33
AD11/A11	35	D_8	34
AD12/A12	36	D_7	35
AD13/A13	37	C_8	36
AD14/A14	38	C_7	37
AD15/A15	39	B_8	38
PC0	40	B_7	41
PC1	41	A_7	42
PC2	42	B_6	43
A19/$\overline{CSI}$	43	A_6	44
V_{CC}	44	B_5	45

Figure 12

EXEMPLES D'APPLICATIONS

La souplesse d'adaptation de ces composants est telle (par définition !) que chaque application est un cas particulier. Pas vraiment au niveau matériel, où il suffit pratiquement d'interconnecter le processeur, son périphérique programmable, et le "monde extérieur", mais essentiellement au niveau de la programmation du PSD 3XX.
Les exemples de schémas qui vont suivre illustrent donc surtout les façons de relier les PSD 3XX à différentes unités centrales, et pourront donc être communs à de multiples applications.
A vrai dire, on pourra complètement transformer un système par simple reprogrammation de l'EPROM et du réseau programmable de son PSD 3XX !

Commençons donc par deux exemples particulièrement typiques d'adaptation des PSD 3XX sur des microcontrôleurs très populaires : le 80C31 à la **figure 13**, et le 68HC11 à la **figure 14**.

Dans un cas comme dans l'autre, les deux ports d'entrée-sortie servant à la communication avec le PSD 3XX sont reconstitués, tandis que ce périphérique programmable apporte ses vastes ressources en mémoire RAM et EPROM (protégée).

Tout cela moyennant un montage guère plus compliqué ou encombrant que s'il s'agissait d'ajouter une simple EPROM !
Passons maintenant aux microprocesseurs, dépourvus par définition de tout port d'entrée-sortie. Les PSD 3XX s'interfacent

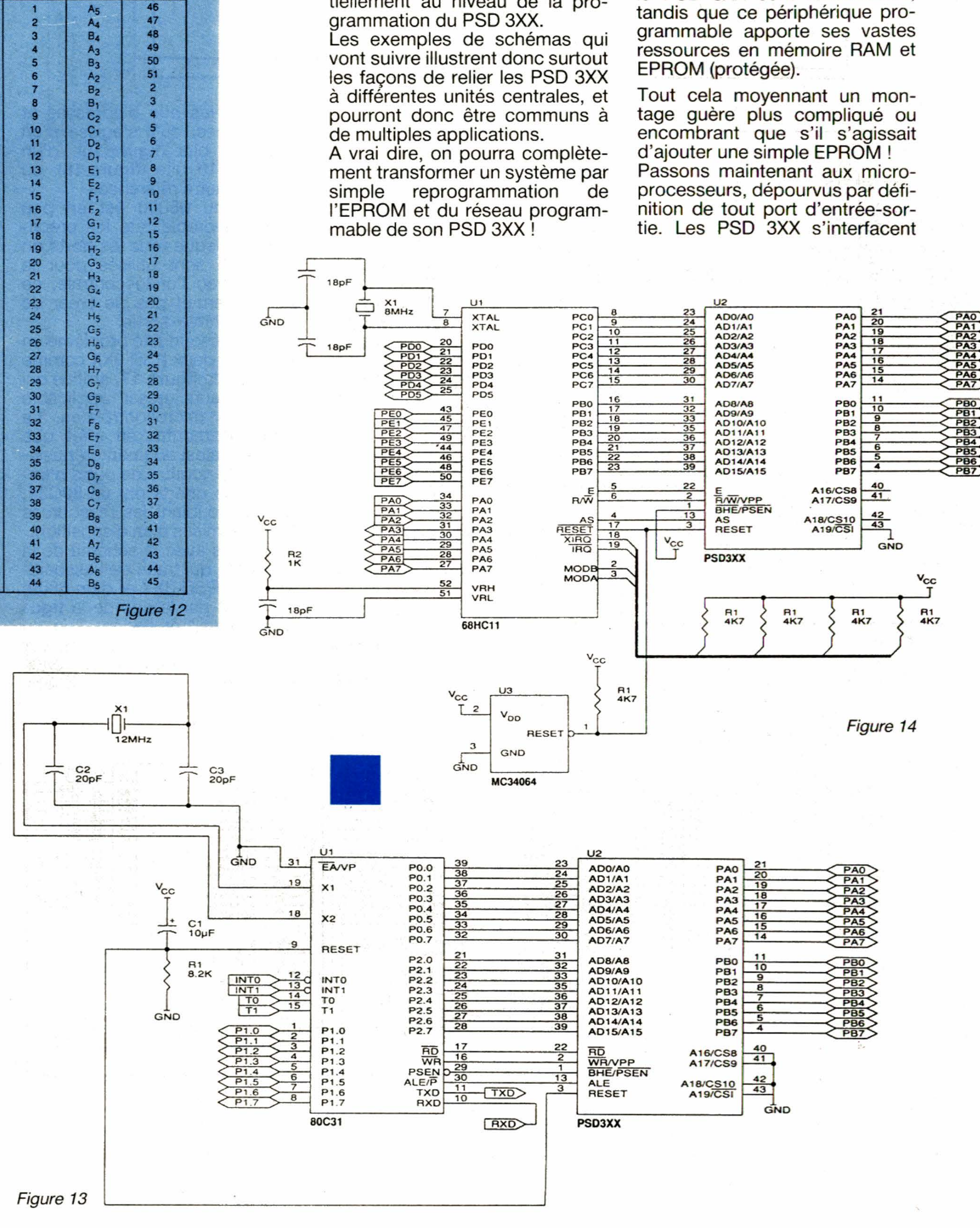


Figure 14

Figure 13

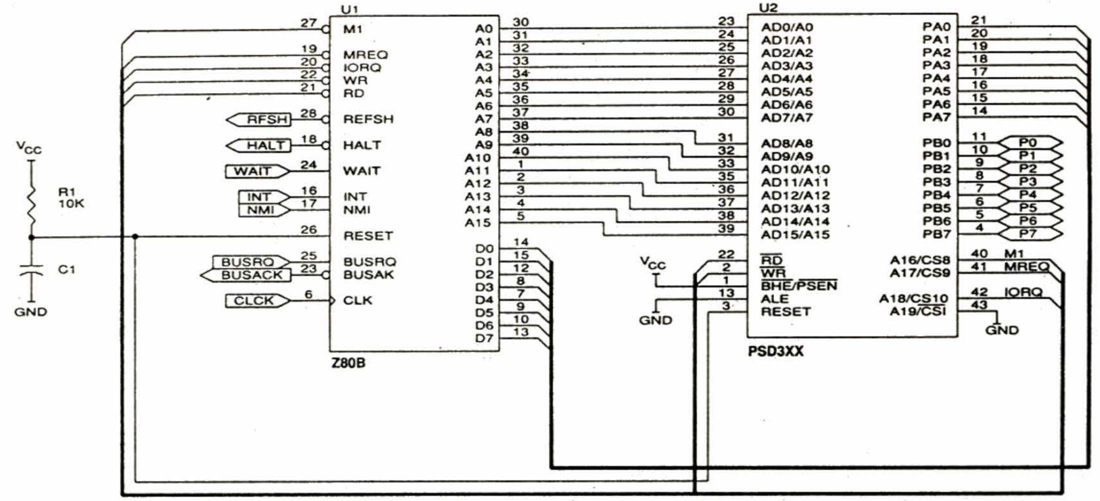


Figure 15

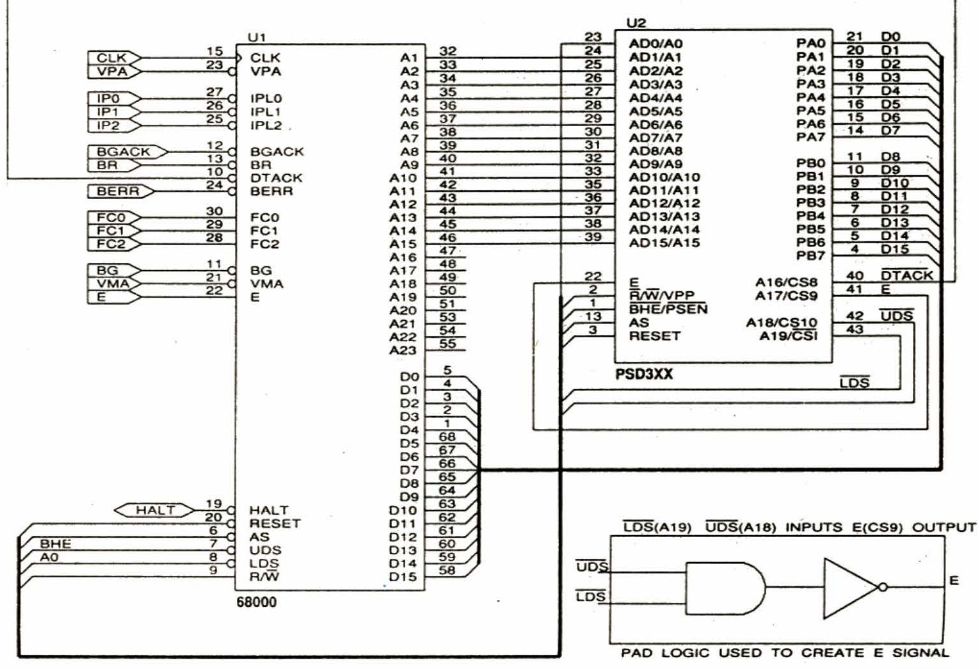


Figure 16

alors sur les bus d'adresses et de données, qu'il s'agisse d'un Z80 à 8 bits (**figure 15**), d'un 68000 à 16 bits (**figure 16**), ou de toute autre référence.

Le cas du 68000 est un peu moins favorable que les précédents, puisque tous les ports du PSD 3XX sont utilisés pour la liaison avec l'unité centrale, ce qui ne permet pas de créer de lignes d'entrée-sortie.

On peut résoudre le problème en associant deux PSD 3XX comme indiqué à la **figure 17**, artifice qui est d'ailleurs aussi applicable avec des microcontrôleurs. La **figure 18** montre en effet que l'on peut ainsi augmenter massivement le nombre de ports d'entrée-sortie offerts par un 80C31, tout en doublant les ressources mémoire mises à sa disposition.

Terminons avec le schéma de la **figure 19**, qui traite de l'association d'un PSD 3XX et d'un 6809, et surtout par le cas de la **figure 20**, détaillant l'adaptation du PSD 3XX sur un microprocesseur à bus multiplexé, en l'occurence un 80186.

Figure 17

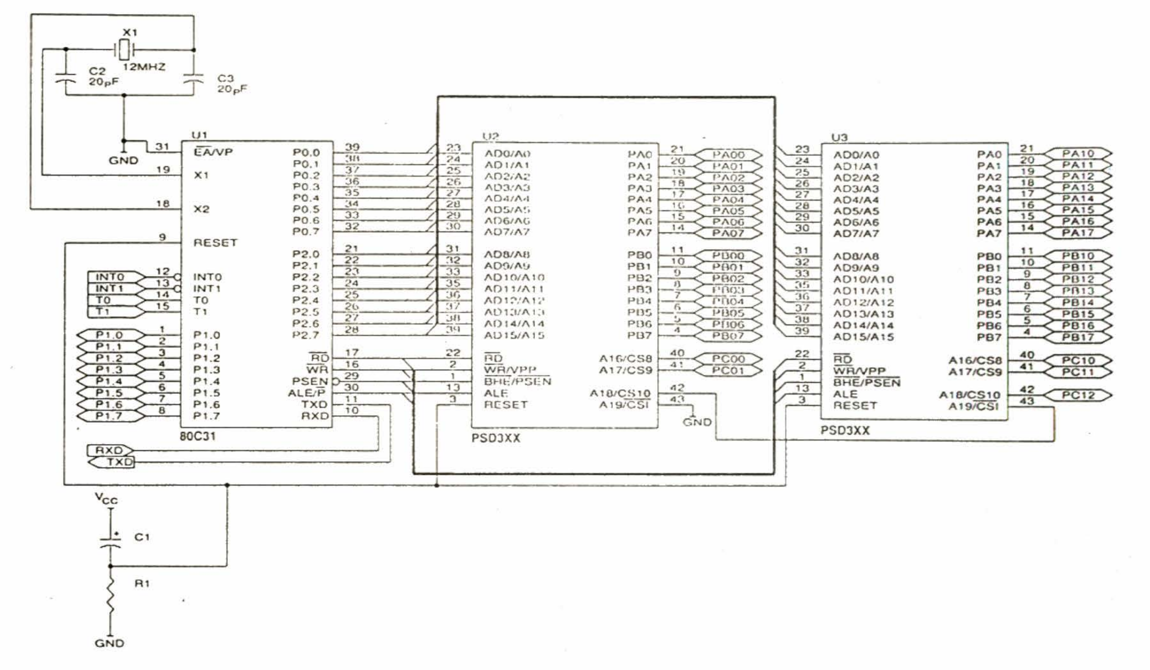


Figure 18

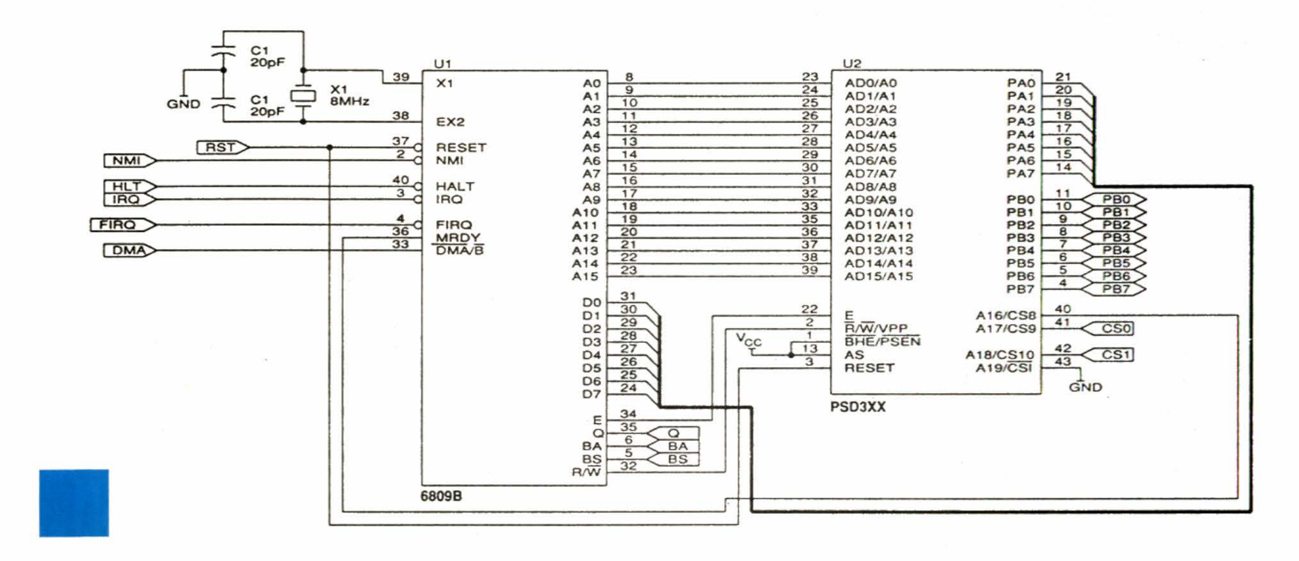


Figure 19

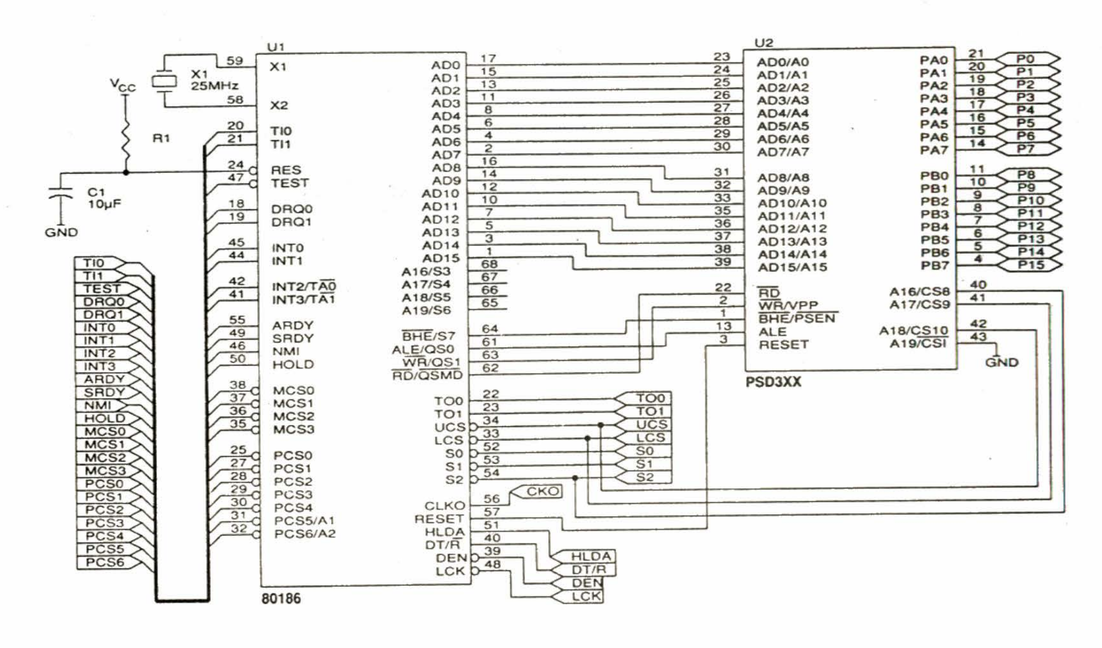


Figure 20

LOGICIELS ET SYSTEMES DE DEVELOPPEMENT

Les PSD 3XX sont bien plus que des EPROM ou des PAL, et nécessitent pour leur mise en œuvre des outils de développement beaucoup plus sophistiqués qu'un éditeur de fichiers et un programmateur même dit "universel".

Votre premier logiciel de développement de PSD 3XX, "tournant" sur le premier PC venu, peut être obtenu sans frais car il fait partie du kit d'évaluation disponible sur demande qualifiée.

Sur une disquette 5" 1/4 haute densité ou trois disquettes de 360 K est ainsi offert un puissant logiciel permettant déjà de mettre intégralement sur pied des applications.

La procédure est particulièrement conviviale, puisqu'il s'agit essentiellement d'un échange de questions et de réponses en langage clair (anglais).

Il faudra cependant remplir quelques tableaux, à commencer par celui décrivant le plan mémoire que l'on souhaite instaurer. A ce stade, une parfaite connaissance du projet en cours et des composants utilisés est indispensable : pas question de répondre à peu près ou de "faire l'impasse" sur des demandes dont on n'est pas certain d'avoir bien compris le sens.

Tout comme l'écriture d'équations pour un compilateur de PLD, c'est véritablement là un travail de développeur averti et bien documenté.

Cette première étape produit un fichier disque qu'il faut encore compiler en un fichier "HEX" exploitable par le programmateur : soit un appareil de la marque, soit un modèle "agréé" (DATA I/O).

Le compilateur ne fait pas partie du logiciel d'évaluation gratuit, mais des deux "packages" disponibles moyennant finances : PSD-Silver (logiciel complet sans le programmateur), ou PSD-Gold (logiciel et programmateur "Magic-Pro").

Il est possible de développer avec le seul logiciel d'évaluation, puis de compiler et programmer ultérieurement en achetant la suite ou en s'adressant à un sous-traitant dûment équipé.

Cela mérite assurément d'essayer...

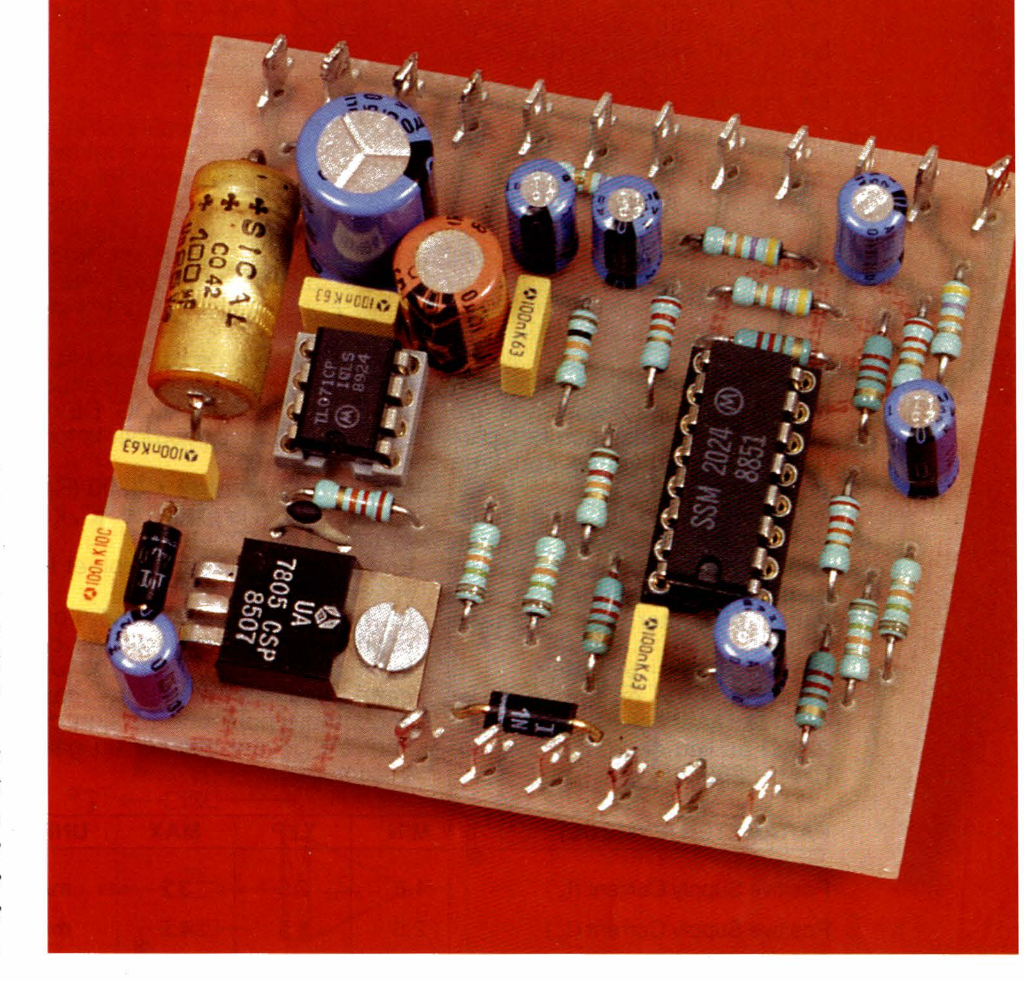
WSI est représenté en France par :

A2M
B.P. 89
78152 LE CHESNAY Cedex
Tél. : (1) 39.54.91.13
Fax : (1) 39.54.30.61

Patrick GUEULLE



Le quadruple VCA SSM 2024



Pour compléter notre collection de circuits SSM, voici cette fois le 2024 : quatre VCAs totalement indépendants, intégrés dans un même boîtier 16 broches. Les performances annoncées par le constructeur ne le mettent pas en tête des VCAs (il n'est d'ailleurs pas dans la liste des produits audio professionnels, mais dans la rubrique "musique électronique"), toutefois il est tellement simple à mettre en œuvre qu'il mérite bien ces quelques lignes car nul doute qu'il aura sa place dans de nombreuses applications.

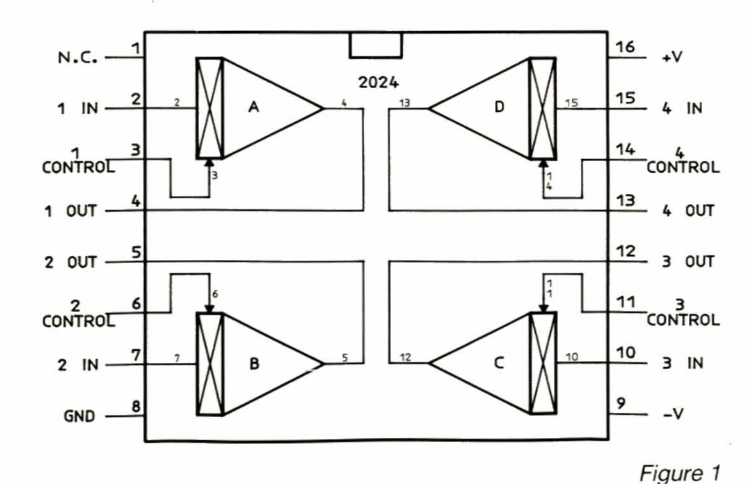


Figure 1

Nous avons constaté avec surprise que notre première rencontre avec les SSM datait déjà de plus de 2 ans !
Aussi profitons-nous de l'occasion pour faire un bref rappel des numéros et des circuits abordés, afin de faciliter vos éventuelles recherches.

N° 509 SSM 1015
N° 510 SSM 2402
N° 512 SSM 2013
N° 513 application 2015/2402
N° 515 SSM 2110
N° 516 SSM 2016 / 2017 / 2142, ainsi qu'une application 2013 / 2110 (limiteur-compresseur / noise gate stéréo).
N° 534 exploitation SSM 2016 (CLEMENT).
N° 535 exploitation SSM 2402 (CLEMENT).

LE 2024

La **figure 1** donne l'organisation

du boîtier de ce circuit, ainsi que la symbolique interne.
Elle permet de constater immédiatement que chaque VCA est réduit à sa plus simple expression : une entrée, une sortie et une broche de commande ! La **figure 2** dévoile les spécifications traditionnelles, et la **figure 3** - importante - établit les relations entre rapport signal/bruit, THD et le niveau crête à crête du signal d'entrée.
En fait on comprend vite que c'est le taux de distorsion que l'on acceptera pour maxi, qui déterminera le niveau d'entrée autorisable. Ainsi, si on tolère une THD maxi de 0,3 %, le rapport signal/bruit sera de 82 dB, mais le niveau d'entrée faible : 40 mV. Pour obtenir les meilleures performances d'offset et réjection de contrôle, il est conseillé de relier les entrées à la masse par une résistance de 200 Ω. Cette résistance fera partie du diviseur d'entrée, constitué de Rin et donc 200 Ω.
Il sera facile de déterminer Rin en fonction du niveau d'entrée choisi (Vin' = 40 mV par exemple), et du niveau que l'on souhaite présenter au montage (Vin) :
Rin = 200(Vin − Vin)/Vin'
Pour Vin = 1V, on trouve Rin = 4800 Ω. Mais si on opte pour 220 Ω au lieu de 200, on obtient Rin = 5280 Ω.

SIGNAL-TO-NOISE
CONTROL REJECTION (MIN)
SIGNAL/NOISE (dB) — CONTROL REJECTION (dB) — INPUT SIGNAL LEVEL (mVp-p)

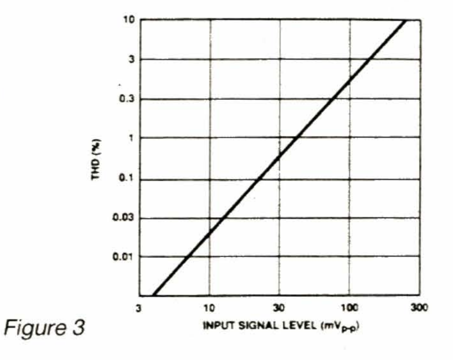


Figure 3

En retournant la question, Vin = (Rin + 200)Vin'/200, on constate que si on a 22 kΩ pour Rin, Vin sera de 4,44 V (ou 4 V pour 220 Ω).
La **figure 4** présente un tracé tenant compte de Iout, Vout (pour Rout, Rin et R control de 10 kOhm).
NOTA : si vous prossédez l'AUDIO/VIDEO reference manual 1992 édité par ANALOG DEVICES, vous pouvez corriger la formule page 7-95 :
Iout = 8,17 Icontrol x ((200/(Rin + 200)) Vin. A la place de "x", il y a "=".
En fait Gm = 8,17 Icontrol, et Io = Gm Vin'.
La figure 4 propose donc trois courbes pour trois valeurs de Icontrol : 100, 200 et 500 µA, cette dernière étant le maxi acceptable pour une tension de contrôle devant assurer un gain maximum.

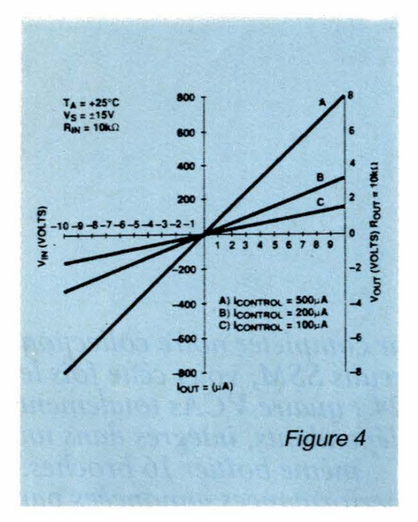


Figure 4

PARAMETER (SYMBOL)	MIN	TYP	MAX	UNITS	CONDITIONS
Positive Supply Current (I_{cc})	1.8	2.5	3.2	mA	I_{CON} (1–4) = 0 V_s = ±15V
Positive Supply Current (I_{cc})	2.5	3.5	4.5	mA	I_{CON} (1–4) = 0 V_s = ±20V
Negative Supply Current (I_{ee})	0.75	1	1.3	mA	I_{CON} (1–4) = 0 V_s = ±15V
Negative Supply Current (I_{ee})	1.04	1.4	1.8	mA	I_{CON} (1–4) = 0 V_s = ±20V
Gain (G)	3842	4085	4330	µmhos	I_{CON} (1–4) = ±500µA
Gain Matching (Δ G)			±5	%	I_{CON} (1–4) = ±500µA
Input Offset Voltage (V_{os})		±.4	±1.5	mV	V_{IN} = 0V I_{CON} (1–4) = ±500µA
Input Offset Voltage (V_{os})		±.4	±1.5	mV	I_{CON} (1–4) = +250µA
Change in Offset Voltage (Δ V_{os})		±100	±400	µV	+2.5µA ≤ I_{CON} (1–4) ≤ +250µA
Change in Offset Voltage (Δ V_{os})		±.25	±1	mV	+250nA ≤ I_{CON} (1–4) ≤ +250µA
Output Leakage (I_{ol})		0.1	±2.5	nA	I_{CON} (1–4) = 0
Control Rejection (untrimmed)	30	41.5		dB	I_{CON}(1–4) = 500µA, V_{IN} (1–4) = 40mV_{pp}
Signal to Noise[1] (S/N)		82		dB	V_{IN} (1–4) = 40mV_{pp}
Distortion[1] (THD)		.3		%	V_{IN} (1–4) = 40mV_{pp}
Threshold Input Control Voltage (V_{TCI})	+160		+220	mV	I_{OUT} (1–4) = 0

Figure 2

L'entrée de contrôle est à basse impédance (environ 250 Ω) et une résistance doit bien évidemment être mise en série si on veut commander en tension.

Un report à la figure 2 montre que le "mini" est obtenu pour une tension comprise entre + 160 et + 220 mV. Toutefois, il faut savoir que si l'entrée de contrôle est portée à 0 V ou − Vcc, le 2024 ne subira aucun dommage. NOTE personnelle de l'auteur : sur un câblage en l'air d'essai (il est vrai, un peu brouillon...) une auto-oscillation s'imposait en dessous de + 100 mV. Attention donc au dessin du CI.

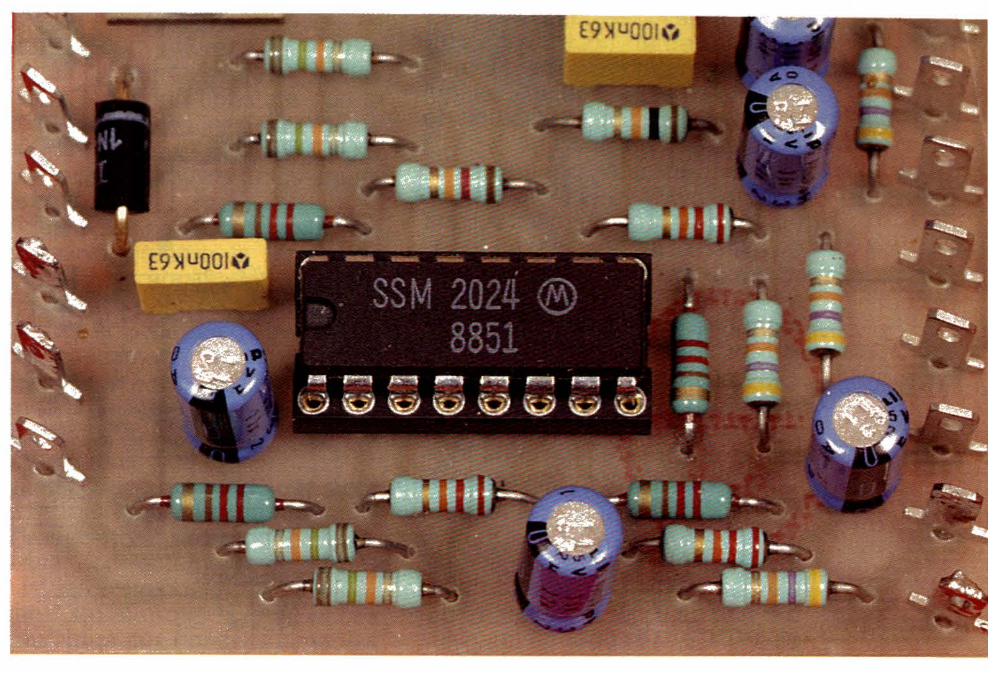
Par contre, si la tension de contrôle est portée à + Vcc, la capacité de dissipation du boîtier ne permet pas de tenir plus de 10 à 20 secondes. Ne pas dépasser + 10 V est conseillé par le constructeur, pour une alimentation + 15/− 15 V (mini +/− 9 V, maxi +/− 18 V).

Les sorties des "cellules" étant en courant, il est possible de les mélanger suivant le principe dit "VE", ou Virtual Earth : masse virtuelle.

Le 2024 ne pouvant délivrer une excursion supérieure à +/ − 500 mV, il faudra - si c'est insuffisant - faire appel à un ampli courant-tension externe.

La **figure 5** donne un exemple "brut" de mélange de 4 voies vers une : difficile de faire plus simple !

Avant de mettre en pratique le 2024, nous vous livrons trois idées de mise en application suggérées par le constructeur : **figure 6** un oscillateur modulé de trois façons, **figure 7** un VCA soumis à une fluctuation externe, et **figure 8** une idée de transit d'une voie vers l'autre plus MASTER (avec un creux central de 6 dB).



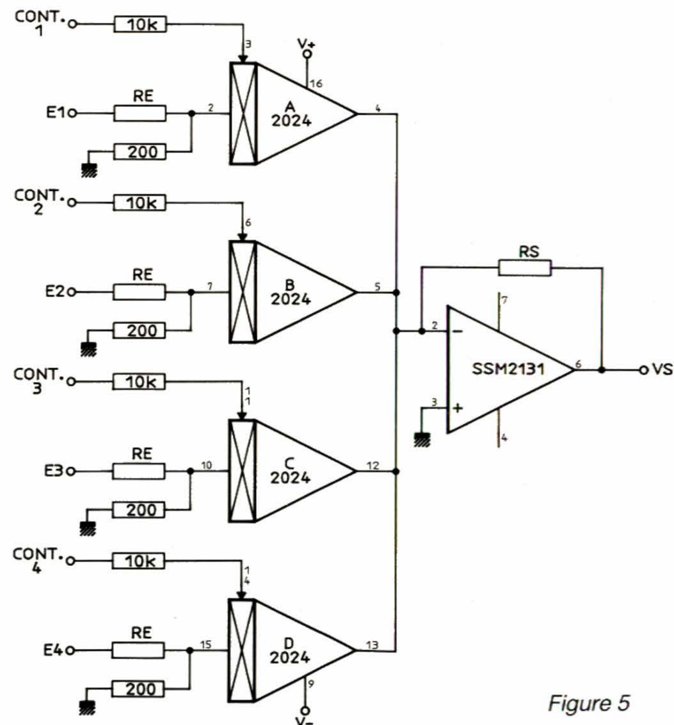


Figure 5

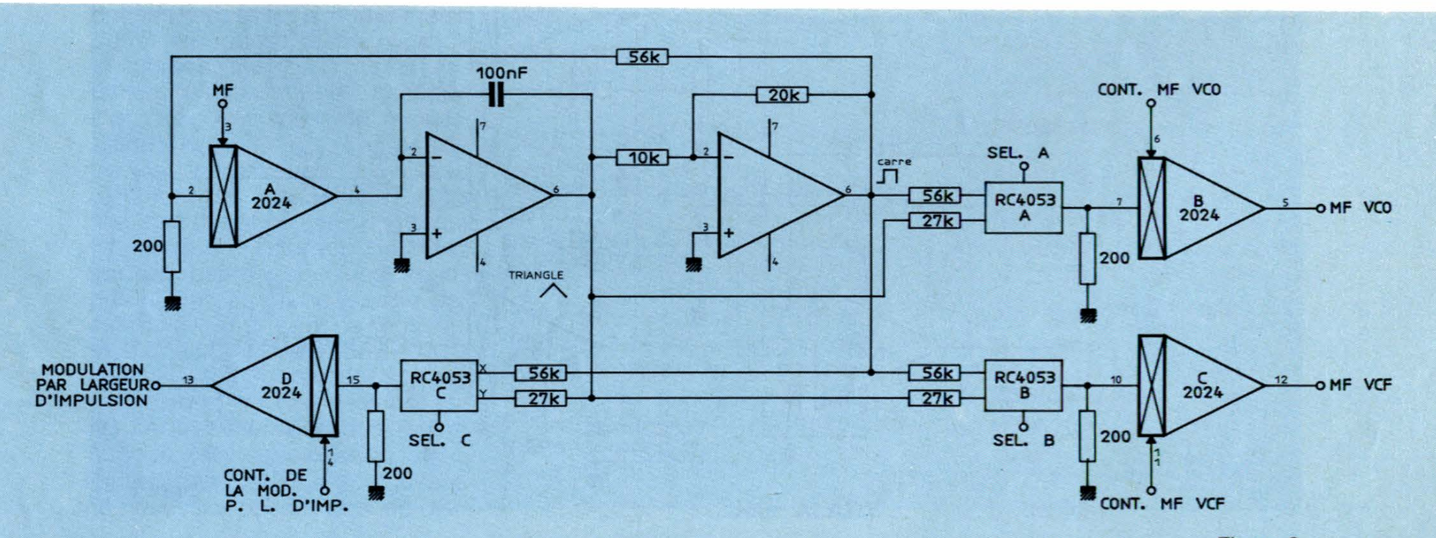


Figure 6

Ce sont des idées à suivre, mais pour notre part nous avons préféré un montage plus "basic" comme indiqué **figure 9**, et qui n'est autre qu'une application de la figure 5.

Les entrées sont prévues pour des niveaux élevés (4 Vc à c), et les broches de contrôle pourront travailler sous 5 V, 300 µA environ pour le "maxi". Afin de faciliter les manipulations, la maquette est équipée d'une réf + 5 V.

Les organes de commandes pourront être alors soit des potentiomètres externes à la carte, ou encore des tensions logiques TTL, des convertisseurs D/A, etc.

Par exemple, il serait envisageable, sur 4 bits, de piloter un VCA par pas de 3 dB, et ce sur 42 dB plus "OFF". On pourrait imaginer ainsi un équaliseur + 20/ – 20 dB, quatre voies (pour un seul 2024), commandé par ordinateur, pour peu que les quatre entrées soient alimentées par des filtres passe-bas, passe-bande(s) et passe-haut.

REALISATION

C'est un bien grand mot pour ce petit montage d'évaluation dont le circuit imprimé est donné **figure 10**. C'est par coquetterie que nous avons mis R8 et R9 en série : juste pour éviter un strap !

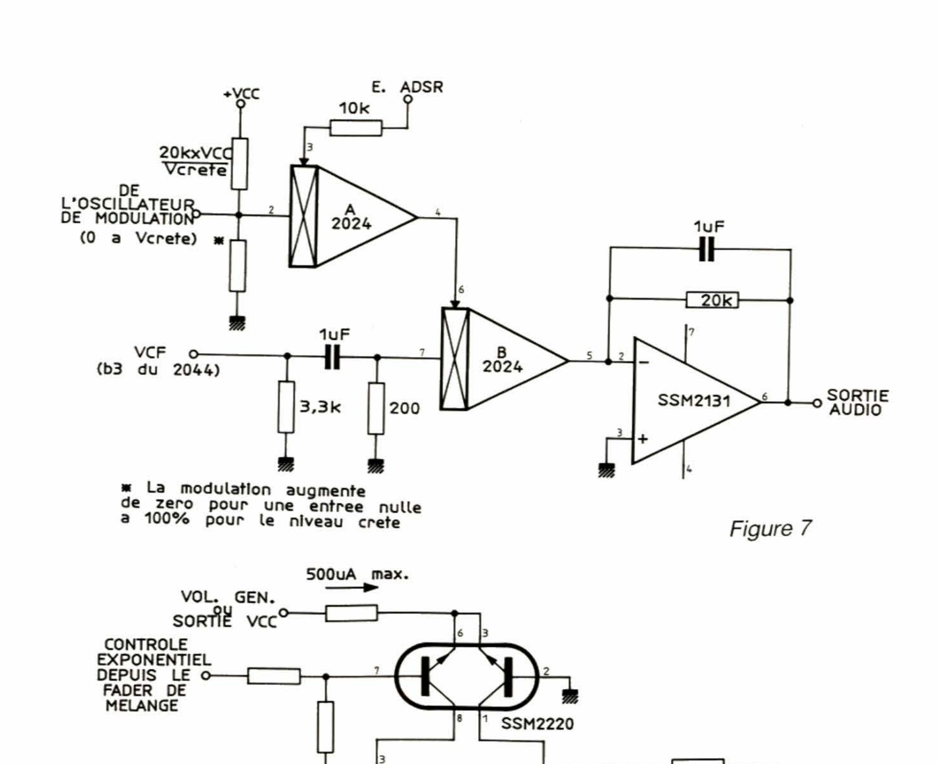


Figure 7

Figure 8

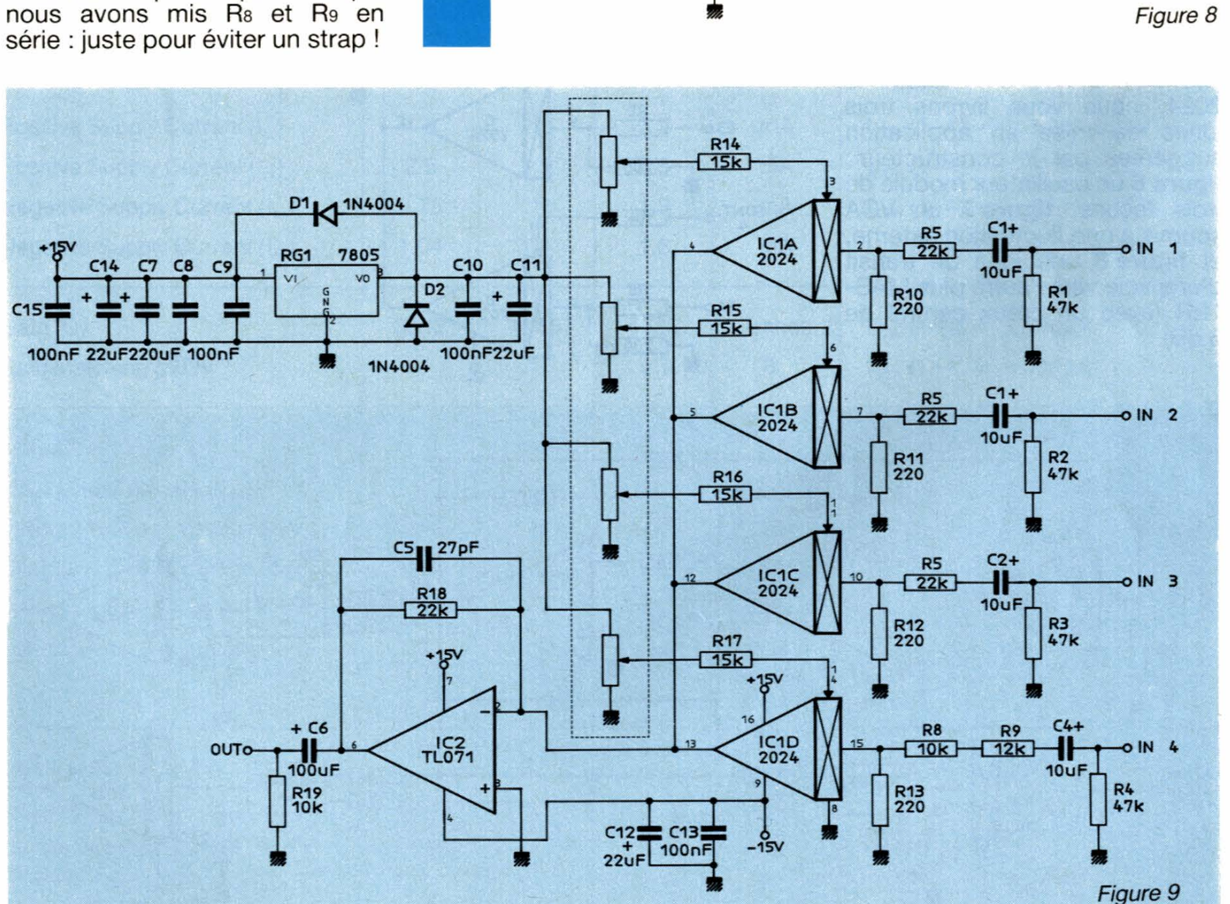
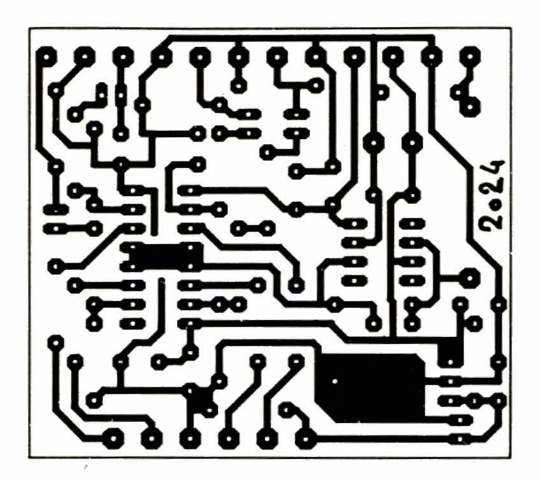


Figure 9



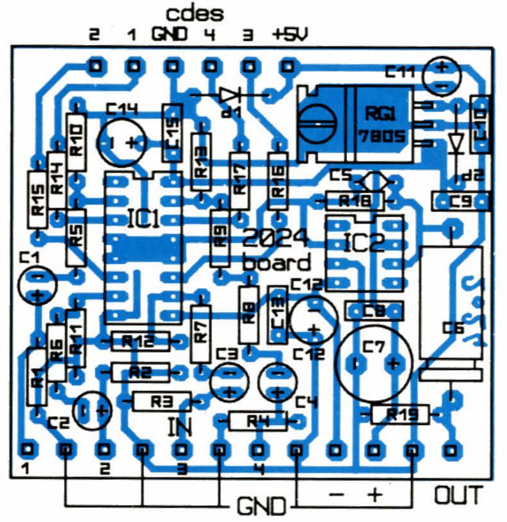


Figure 10

Il n'y a rien à ajouter, ni pour la construction, ni pour la mise en route. Cette "base" permettra de faire divers essais et mesures dans de bonnes conditions, et de constater que le 2024 n'est pas sans intérêt. Le faible niveau d'entrée nous avait fait un peu peur au départ, mais les résultats ont balayé nos craintes.
Il ne serait pas très sérieux de l'utiliser comme élément de groupe dans une console haut de gamme, mais il y a mille applications possibles dans des domaines aussi divers que la téléphonie (insert), la communication (intercom), la musique électronique qu'elle soit de synthèse ou non, etc. Le fait de pouvoir commander 4 voies à distance par un câble à 6 fils seulement - et ce sans risque de "ronflette" - pourrait simplifier parfois certaines télécommandes sur un ampli de guitare par exemple. En effet, il ne faut pas se bloquer sur notre carte d'évaluation : nous y avons sommé les voies, mais rien n'interdit de les commander individuellement sur des trajets très divers.
Régler la saturation d'une guitare ne nécessite pas un taux de distorsion extrêmement faible - vous en conviendrez -, idem pour certains autres effets (delay etc...). Aussi on constate que le 2024 est en mesure de rendre de très nombreux services, pour une mise en œuvre ridiculement simple.
Fidèle à ses habitudes, votre serviteur va prendre le temps de choisir une application utile (ou amusante) pour ce circuit SSM, et nous en reparlerons bientôt.
Soigneusement cachées dans des boîtes à malice, il nous reste encore quelques merveilles de la marque à vous faire découvrir, et pas les moindres !
Restez donc à l'affût, car pour une fois que des composants d'excellente qualité sont bien diffusés, correctement distribués par les revendeurs qui "bougent", que nous avons accès pour vous aux conseils aimables d'ANALOG DEVICES (dont les DATA B. sont mis à jour régulièrement), c'est exceptionnel !
Et les constructeurs de matériels audio ne s'y sont pas trompés : dans SONO d'avril 92, Monsieur LEMERY attirait l'attention du lecteur (page 98) sur le fait que la console ALLEN & HEATH SCEPTER RACK était équipée de 2142 en sorties et de 2017 en entrées ; et que c'était "une première". Puis de préciser pour le 2017 que le constructeur en commandait le gain par un potentiomètre !
Si vous avez lu attentivement le numéro 516 d'ERP, votre sang n'a du faire qu'un tour : quelle est donc alors la courbe de ce potentiomètre ? Comme notre confrère ne le précisait pas, nous avons regardé la photo de façade : ouf, la gravure était linéaire, de 0 à 10 !
Ce n'est pas une critique : juste une constatation "mécanique", mais si on lit ERP, on en sait un peu plus sur les circuits SSM qui vont certainement s'imposer de plus en plus dans les matériels grand-public.

CONCLUSION

Nous terminerons pour cette fois par une idée : le 2024 pourrait parfaitement servir à commander le threshold dans la side chain d'un compresseur, limiteur ou noise-gate comme ceux que nous vous avons jadis proposés.
Sur notre carte d'évaluation nous avons mesuré en effet un affaiblissement de 75 dB, ce qui est tout à fait performant.
Amusez-vous bien !

Jean ALARY

Nomenclature

Résistances

R_1 à R_4 : 47 kΩ
R_5 à R_7 : 22 kΩ
R_8 : 10 kΩ
R_9 : 12 kΩ
ou R_8 : 22 kΩ et R_9 : Strap
R_{10} à R_{13} : 220 Ω
R_{14} à R_{17} : 15 kΩ
R_{18} : 22 kΩ
R_{19} : 10 kΩ

Condensateurs

C_1 à C_4 : 10 µF 63 V radial
C_5 : 27 pF
C_6 : 100 µF 25 V axial
C_7 : 220 µF 25 V radial
C_8, C_9, C_{10}, C_{13}, C_{15} : 0,1 µF Milfeuil
C_{11}, C_{12}, C_{14} : 22 µF 63 V radial

Semiconducteurs

IC_1 : SSM 2024
IC_2 : TL071
RG_1 : 7805
D_1, D_2 : 1N 4004

S.N. RADIO PRIM A DÉMÉNAGÉ !

Ouvert du lundi au samedi de 9 h 30 à 12 h 30 et de 14 h à 18 h 45. Fermé le dimanche.

FLUKE® PHILIPS

FLUKE 87 Affichage analogique/numérique 4 000 points. Sélection automatique de gamme. Calibre : 400 mV à 1 000 V DC, 400 mV à 1 000 V AC, 400 µA à 10 A DC, 400 µA à 10 A AC, 400 à 40 Mohms.

PLUS : capacimètre, compteur de fréquence, enregistrement min/max, crête min/max.

PRIX : **3167F TTC**

SCOPE METER

Oscillosco...

CONVERTISSEUR STATIQUE DE POCHE

Entrée de 10 à 15 V continu par fiche allume-cigare - Sortie : 220 V 50 Hz alternatif sinusoïde modifié - Fiches américaines ou françaises - Régulation : tension ± 5 % - Fréquence ± 1 % - Puissance de sortie : 100 W max - Rendement : 90 % - Courant à vide : 0.08 Amp.

APPLICATIONS

TV et magnétoscopes - Equipement audio - Lampes et néons - Maintenance électronique - Equipement domestique : ventilateur, rasoir, etc. - Chargeur de batteries Ni Cad - Micro-informatique

1485F TTC

CONVERTISSEURS À TRANSISTOR...

DC/220 V A...

OUVERT TOUT L'ÉTÉ

NOUVELLE ADRESSE
S.N. RADIO PRIM

159, rue Lafayette - 75010 PARIS

Tél. : (1) 40.35.70.50 - Fax : (1) 40.35.43.63

(à l'angle du 1 rue de l'Aqueduc, à 50 mètres de l'ancienne adresse sur le même trottoir).

HORAIRES D'ÉTÉ : Juillet, du lundi au samedi de 9 h 30 à 12 h 30 et de 14 h à 18 h 45. Août, du mardi au samedi de 9 h 30 à 12 h 30 et de 14 h à 18 h 45.

Métro : Gare du Nord ou Gare de l'Est

...avec boîtier isolé. **510F**

...201. PUissance 225 W.
- Entrée sur fil souple, 12 V DC
- Sortie 220 V AC 50 Hz ± 5 % sur douilles Ø 4 entre axe 19 mm.
- Présentation : identique au CV 101
- Les transistors sont montés sur des radiateurs en profilé d'aluminium.
- Dimensions : L 140 × H 110 × P 167
- Poids : 5.5 kg.

747F

CV 201 I Idem caractéristiques CV 201 mais avec boîtier isolé **875 F**

...posants ...passifs. Mesure. Outillage. Connectique. Transformateurs. Coffrets. Etc.

25F franco de port (Remboursé à la 1re commande de 150F minimum)

BON DE COMMANDE ERP 07/92

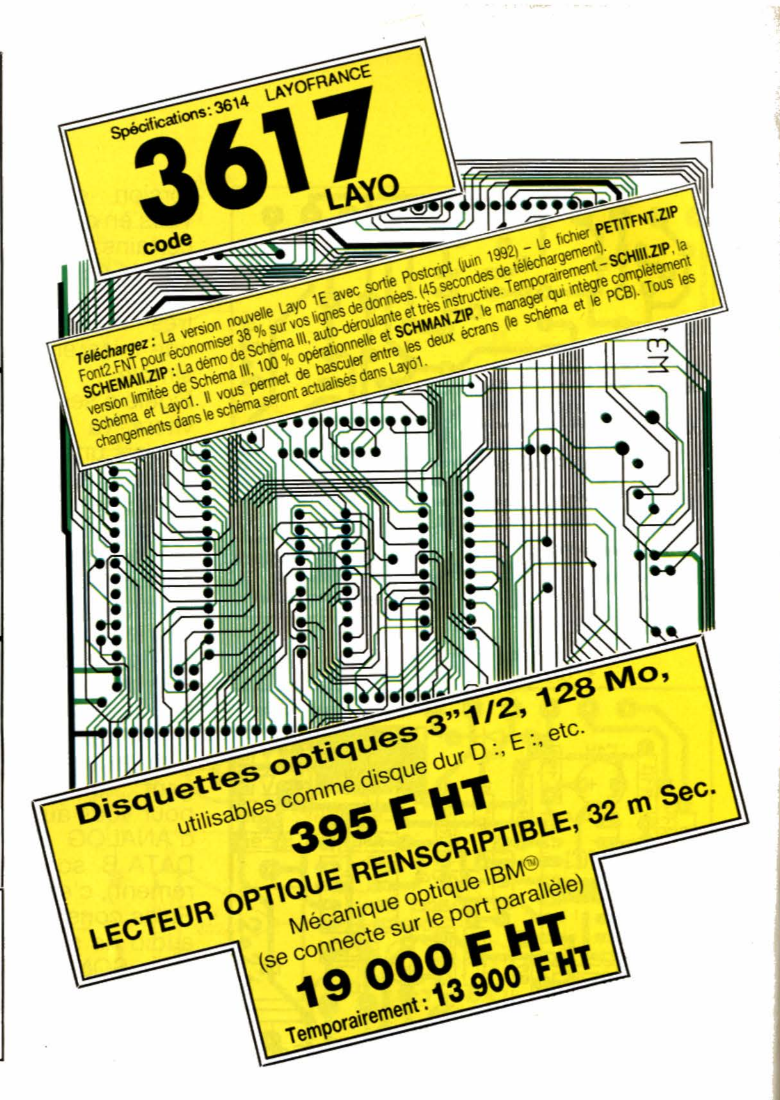
Veuillez me faire parvenir le catalogue 92, ci-joint **25F** en chèque à l'ordre de **S.N. RADIO PRIM**

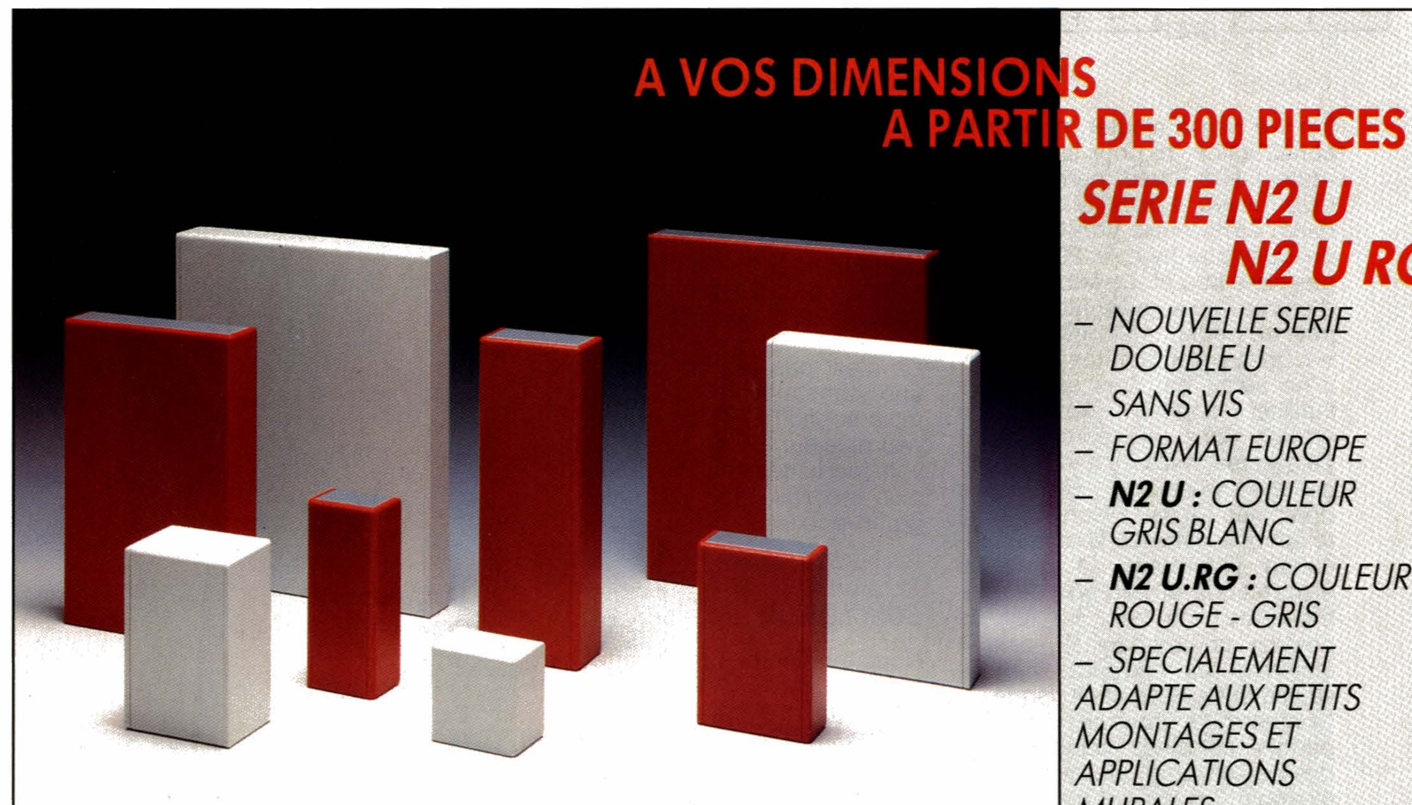
NOM ....

Prénom ....

Adresse ....

CP | | | | | | Ville ....



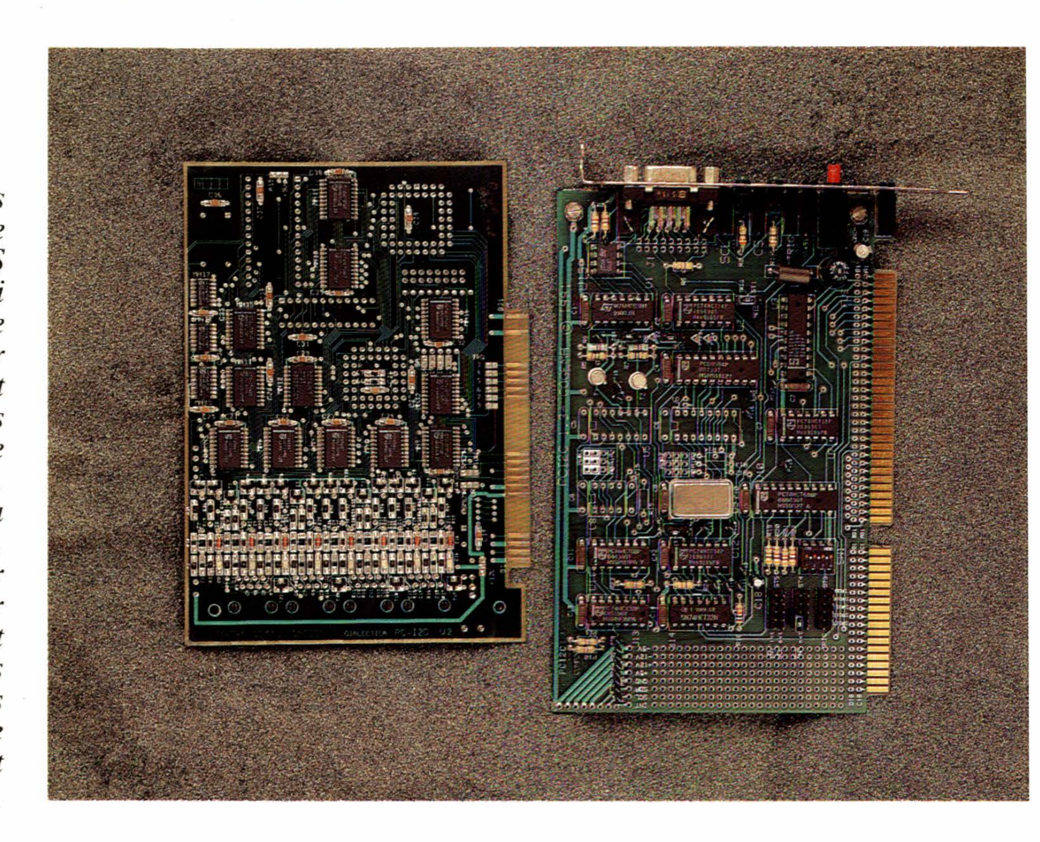


N2 U1 : 25 x 40 x 40
N2 U2 : 20 x 90 x 35
N2 U3 : 25 x 53 x 163
N2 U4 : 25 x 53 x 83
N2 U5 : 35 x 53 x 85
N2 U6 : 20 x 103 x 163
N2 U7 : 20 x 163 x 203

DEPARTEMENT : PRODUITS STANDARDS
LA TOLERIE PLASTIQUE
Z.I ROUTE D'ETRETAT
76930 OCTEVILLE/MER
Tél. : **35.44.92.92**
Fax : **35.44.95.99**

Circuit d'interface bus 8 bits parallèle Bus I2C

Nous allons aujourd'hui vous entretenir du circuit PCD 8584 de PHILIPS SEMICONDUCTORS qui rendra service à beaucoup d'entre vous qui, lisant nos articles sur l'I2C depuis longtemps et utilisant pour des raisons historiques d'autres microcontrôleurs que ceux de la famille 80C51, voudraient se raccrocher au peloton de notre grande famille. Ce circuit PCD 8584 a pour fonction principale de réaliser l'interface "hard I2C" pouvant être commandée par un bus parallèle conventionnel 8 bits provenant soit de microcontrôleurs ordinaires soit de bus.



Vous allez certainement nous faire les remarques suivantes :
– Pourquoi avoir attendu si longtemps pour nous le présenter ?
– Pourquoi d'autres revues en ont elles parlé avant vous en présentant quelques schémas d'applications ?
Et vous aurez raison.
En fait nous avons attendu un long moment avant de vous proposer ces lignes afin de posséder suffisamment d'informations sur ce circuit pour vous décrire en profondeur la quasi totalité de ses ressources et elles sont nombreuses.
Qui sait attendre... et nous l'espérons, votre patience sera récompensée.

LE PCD 8584 ET SA VIE INTERNE

Pour une fois nous avons décidé de vous présenter ce circuit d'une manière non conventionnelle.
Il était une fois, il y a bien longtemps PHILIPS avait développé une famille de microcontrôleurs à cœur de 8048 (cela ne nous rajeunit pas !) portant le nom de 84 xxx en technologie NMOS et 84C xxx en SACMOS et comportant déjà une interface I2C en "hard".

Ces interfaces, bien que moins performantes que ceux des 80C 552 ou 652 apparus bien plus tard (que nous avons décrits récemment) fonctionnaient parfaitement (maître, esclave, mode mulitimaster et tutti quanti...).

Le marché aidant, une demande grandissante de circuits interface I2C vit le jour et fit songer à concevoir le PCD 8584. Il devenait alors très facile d'effectuer le découpage de la partie I2C des 84(C) xxx afin de la mettre dans une petit boîtier et de lui adjoindre tout ce qui faudrait pour l'interfacer aux bus 8 bits de la plupart des microcontrôleurs du commerce (8048, 80(C) 51, 68000, Z80, ...) en laissant en plus toutes les facilités conventionnelles d'usage — fonctionnement par interruption ou par polling — puis en ajoutant quelques gadgets pour les connaisseurs, du style "strobe", "mode moniteur", "mode longue distance"...

Figure 1

Ceci permet bien sûr de se raccrocher à des bus internes de systèmes quelconques (bus interne de PC par exemple...).
Les **figures 1** et **2** vous indiquent le brochage et la configuration interne de ce circuit. Bien qu'il ne soit pas dans nos habitudes de vous commenter le brochage des circuits, il est aujourd'hui important de l'analyser plus finement, ce qui va nous permettre d'évacuer certaines des questions qui nous sont fréquemment posées.

Les broches DB_0 à DB_7

Ce sont simplement les broches d'entrées/sorties parallèles des "données".

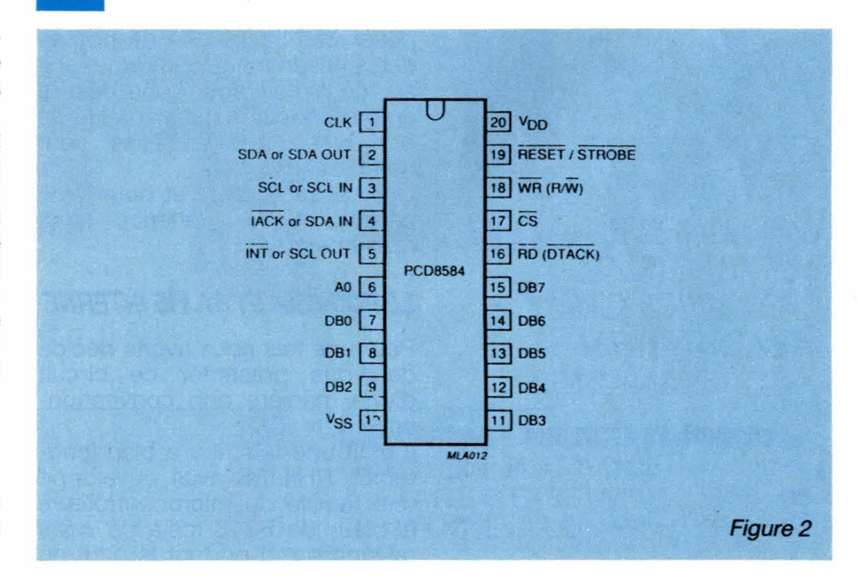


Figure 2



CLK

C'est la broche d'entrée du signal d'horloge dont le circuit a besoin pour créer le signal d'horloge "SCL" de l'I2C (ainsi que les signaux internes d'échantillonnage nécessaires au filtrage numérique des SDA et SCL de l'I2C).
Le circuit n'ayant aucun sens divinatoire, il lui est impossible de deviner quelle est la valeur de fréquence du signal d'horloge que vous lui avez appliqué sur sa broche et prend par défaut f(CLK) = 12 MHz.
Si, pour des raisons qui vous seraient propres, vous désirez utiliser d'autres fréquences, vous devrez donc dans ce cas le lui faire savoir en chargeant dans l'un de ses registres internes (S_2) certaines valeurs à des bits précis (S_{22}, S_{23}, S_{24}, tableau **figure 3**), de façon à ce qu'il confi-

bit			clock frequency
S24	S23	S22	(MHz)
0	X	X	3
1	0	0	4,43
1	0	1	6
1	1	0	8
1	1	1	12

Figure 3

gure ses petits prédiviseurs pour espérer pouvoir vous comprendre. Ceci fait, et par la même occasion, on choisit la valeur désirée de l'horloge du "SCL" du bus I2C et l'on positionne les bits S_{20} et S_{21} en accord avec le tableau de la **figure 4** et enfin on

bit		SCL approximate frequency
S21	S20	(kHz)
0	0	90
0	1	45
1	0	11
1	1	1,5

Figure 4

se sent à peu près tranquille avec toutes ces horloges.
Venons en maintenant aux complications.
A l'exception du /CS qui sert à mettre en branle le circuit, toutes les autres broches semblent pouvoir poser des problèmes ! Pour éviter cela un retour aux sources est nécessaire.
Deux grandes parties sont à distinguer sur la **figure 1**, celle du bas qui a pour but d'effectuer l'interfaçage entre les différents types de bus parallèles et le jeu de registres internes S_0, $S_{0'}$, ... S_3.
Summun du vice, l'interface du bas a des relations intimes avec le contenu des registres.
Comment tout cela fonctionne-t-il ? Essayons donc de commencer par le début.
Lors de la première mise sous tension (ou bien sur un reset externe volontaire via la broche RESET), sans autre forme de procès, le circuit se croit "jeté" dans un système équipé d'un microcontrôleur de la famille 80C51 (option par défaut). Comme de toute façon il faut bien un début à tout, la toute première chose qu'il est OBLIGATOIRE d'effectuer avec ce circuit est de lui déclarer comment il doit s'appeller du point de vue I2C et donc d'écrire le nom de cette adresse dans le registre S_0. Or, oh miracle en désirant faire cela, le microcontrôleur implanté sur votre système (jusque là inconnu du PCD 8584) produira des signaux qui lui sont spécifiques afin d'écrire le fameux registre $S_{0'}$, et le PDC 8584 qui a été éduqué pour reconnaitre tout ce petit monde se mettra à cette occasion automatiquement sur la bonne configuration de microcontrôleur.
Le tableau de la **figure 5** rappelle

Les deux autres ont des fonctions doubles :
S_0 : un registre tampon de données et de décalage
S_1 : un registre de commande et de status
De plus, ces derniers, utilisés pendant les échanges (transmission ou réception), peuvent être séparément écrits ou lus.
Reprenons en détail le contenu de ces registres.

$S_{0'}$: Bien évidemment ce n'est que dans le cas où le PCD 8584 est "esclave" que l'adresse I2C servira mais, comme nous vous l'avons indiqué, quoi qu'il en soit, il est nécessaire de commencer par charger une valeur d'adresse (une valeur sur 7 bits, le bit de poids fort comptant pour du beurre). A noter, au reset, la valeur interne est "00" en hexa.

Remarque très très importante :

Si l'on charge la valeur "x001 0000" (10 hex) dans $S_{0'}$ en pensant que 10 hex sera l'adresse "esclave" I2C du composant, c'est FAUX !
En effet, lorsqu'un message incident se présente, il entre en S_0 et sa valeur (sur 8 vrais bits, ex. :

Type	R/W	WR	RD	DTACK	IACK
MAB8049/51	NO	YES	YES	NO	NO
SCC68000	YES	NO	NO	YES	YES
Z80	NO	YES	YES	NO	YES

Figure 5

les différents types de signaux nécessaires et utilisés par le PCD 8584 lors d'échanges selon les différents types de microcontrôleurs, montrant ainsi que le ménage est facilement réalisable automatiquement.
Nous voici débarrassés de quelques broches de plus.

Les registres internes et comment y accéder

Le PCD 8584 comporte 5 registres. Trois d'entre eux sont utilisés pour l'initialisation du circuit. Normalement, ils sont écrits juste après le RESET du circuit. Ce sont :
$S_{0'}$: propre adresse I2C du PCD 8584 (le premier à écrire !)
S_2 : le registre définissant les "horloges" (déjà évoqué)
S_3 : le vecteur d'interruption (voir plus loin)

adresse + R/W), afin d'être intelligemment comparée, remplit intégralement le registre associé de comparraison entre S_0 et $S_{0'}$. Or, à cet instant, le bit de poids fort est significatif et représente le bit le plus élevé de l'adresse I2C. Autrement dit la valeur inscrite en $S_{0'}$ a été décalée à "gauche" d'un cran au niveau du registre de comparaison et ayant écrit "x001 0000" (10 hex) en $S_{0'}$, le PCD 8584 se reconnait sur x"0010 0000" soit 20 hex, sa véritable adresse.
Lorsque le circuit est appellé et qu'il s'est reconnu, le bit "AAS" (Adress As Slave) du registre de Status S_1 est positionné à 1 pour des emplois ultérieurs.

S_2 : Précédemment nous vous avons déjà défini les 5 bits de poids faibles. Les autres sont inopérants (quelle tristesse, quel gâchis).

S_3 : Ce registre est conçu pour contenir la valeur du vecteur d'interruption afin de pouvoir utiliser des interruptions vectorisées.
La valeur de ce vecteur est présentée sur le port parallèle quand le signal d'acquittement d'interruption est présent et que le signal ENI (ENable Interrupt) est positionné.
Au reset (donc dans le mode 80C xx), sa valeur est "00" hexa et à l'initialisation du mode 68000, la valeur est de "0F".

S_0 : S_0 est une combinaison d'un registre à décalage et d'un registre tampon.
Les données parallèles sont toujours soit écrites vers le registre à décalage, soit lues du registre tampon.
Par contre les données série sont "décalées" (ou en entrée ou en sortie) à l'aide du registre à décalage. Une petite remarque complémentaire est à faire concernant le mode de réception car à ce moment là les données contenues dans le registre à décalage sont recopiées dans le tampon pendant la phase d'acquittement.

S_1 : C'est le plus compliqué et nous vous l'avons réservé pour la fin !
En fait il est dédoublé en deux registres, l'un pour la lecture seule, l'autre pour l'écriture seule, pour tous les bits de l'un et de l'autre.

"En écriture :

Le "nibble" ("quarté" comme ont dit au tiercé !) de poids fort a pour occupation de s'intéresser à valider les entrées/sorties d'informations soit sur les broches directement soit dans les registres internes.
Il est composé du bit ESO (= Enable Serial Output) ayant pour but d'autoriser ou non la disponibilité du bus série I2C et des bits ES_1 et ES_2 dont les fonctions, jumelées à la broche A_0, permettent ou non l'accès au contenu des registres Sx.
Afin d'abréger votre supplice de la litanie du "who is who" de chacun d'entre eux et qui dépasse le cadre de l'article, nous vous renvoyons aux caractéristiques du constructeur. Pour les curieux, les tableaux des **figures 6** et **7** résument leurs fonctions.
Le deuxième "nibble" (poids faible) s'occupe de la gestion de l'information de sortie de l'interruption, des conditions de départ, d'arrêt et d'acquittement du bus I2C. Même motif, même punition, tableau **figure 8**.

A0	ES1	ES1	IACK	Opération
H	X	X	X	READ/WRITE CONTROL REGISTER (S1) STATUS (S1) not available
L	0	0	X	READ/WRITE OWN ADDRESS (S0')
L	0	1	X	READ/WRITE INTERRUPT VECTOR (S3)
L	1	0	X	READ/WRITE CLOCK REGISTER (S2)

Figure 6

A0	ES1	ES2	IACK	Opération
H	X	X	H	WRITE CONTROL REGISTER (S1)
H	X	X	H	READ STATUS REGISTER (S1)
L	X	0	H	READ/WRITE DAT (S0)
L	X	1	H	READ/WRITE INTERRUPT VECTOR (S3)
X	0	X	L	READ INTERRUPT VECTOR (ack nowledge cycle)
X	1	X	L	long-distance mode

Figure 7

STA	STO	present mode	function	operation
1	0	SLV/REC	START	transmit START + address remain MST/TRM if R/W = logic 0 ; go to MST/REC if R/W = logic 1
1	0	MST/TRM	REPEAT START	same as for SLV/REC
0	1	MST/REC	STOP READ	transmit stop
		MST/TRM	STOP WRITE	go to SLV/REC mode
1	1	MST	DATA CHAINING	send STOP, START and address after last master frame without STOP sent
0	0	ANY	NOP	no operation

Figure 8

"En lecture" :

Le registre S_1 en mode de lecture contient bit à bit toutes les informations d'état (STATUS) de ce circuit intégré. Ici aussi la spécification du constructeur est très complète et nous nous contenterons de vous donner quelques exemples :

STS : Ce bit indique la détection d'une condition de STOP lorsque le composant est en mode d'esclave récepteur.

BER : Ce bit est positionné pour indiquer qu'une erreur de protocole a été détectée sur le bus (STOP ou START pas à leur place...).

AAS : Lorsque le composant fonctionne en mode d'esclave récepteur, ce bit indique que le circuit a reconnu sur le bus I2C son adresse.

LAB : En mode multimaster, ce bit indique que le circuit a perdu l'arbitrage.

BB : Indique que le bus est occupé et donc inaccessible momentanément.

PIN : Ce bit est la clef de voute du fonctionnement de ce circuit. En effet c'est sa gestion qui régit l'échange I2C / microcontrôleur. Comme son vrai nom l'indique (!) "Pending Interrupt Not", ce bit a pour mission de vous signaler si il est temps ou non de traiter, par interruption du microcontrôleur, le contenu du circuit.
Nous vous donnerons plus de détails à son sujet, sur la façon de le commander, lors de la partie software.

Comment accéder aux registres ?

Tout ce que nous venons de vous raconter est bien gentil, mais l'un des problèmes que nous avons passé totalement sous silence réside dans le fait de savoir comment aller lire ou écrie tous ces registres.
Nous avons déjà glisser subrepticement quelques mots à ce sujet mais maintenant rentrons dans le détail de ce genre peu commun d'architecture.
Comme vous venez de le comprendre, ce circuit est composé de quelques grands blocs orientés autour du seul élément actif qu'est le registre tampon/déca-

lage S₀. Si vous avez de bons yeux, vous verrez sur la **figure 1** qu'un bus interne 8 bits parcourt l'intégralité du circuit.

Il représente la colonne vertébrale de l'ensemble et c'est par son intermédiaire que tout le monde communique à l'intérieur du circuit. Quant aux ordres de commandes (à quel moment, comment, où...), ils sont obtenus par décodage de signaux externes appliqués à des circuits logiques situés dans le bas de la figure.

il faut aussi garder en mémoire que tout est piloté par le microcontrôleur extérieur via d'une part son bus parallèle (8 bits) qui est appliqué aux entrées DB_0 et DB_7 et d'autre part via des signaux (ou bus) de commande RD/WR... et que c'est donc à ses ordres que le tout doit/va réagir.

Ceci étant, il suffit donc que le microcontrôleur présente 8 bits (en parallèle) sur le port DBx et simultanément (ou parfois légèrement avant) les bons signaux de commandes pour que le PCD 8584 comprenne après décodage qu'il doit écrire ou lire tel ou tel registre. En résumé il se présente comme un "périphérique" particulier du microcontrôleur.

Figure 9

La **figure 9** résume la façon d'accéder à ces registres.

Un dessin étant souvent beaucoup plus clair qu'un long discours, nous vous donnerons dans le prochain article l'ossature complète du style de logiciel qu'il est nécessaire d'implanter pour que le circuit soit piloté correctement, donc un peu de patience.

DES APPLICATIONS

Et elles sont nombreuses...

Avec la famille 80C51

Nous vous renvoyons à la **figure 10** où les synoptique et schéma d'application sont don-

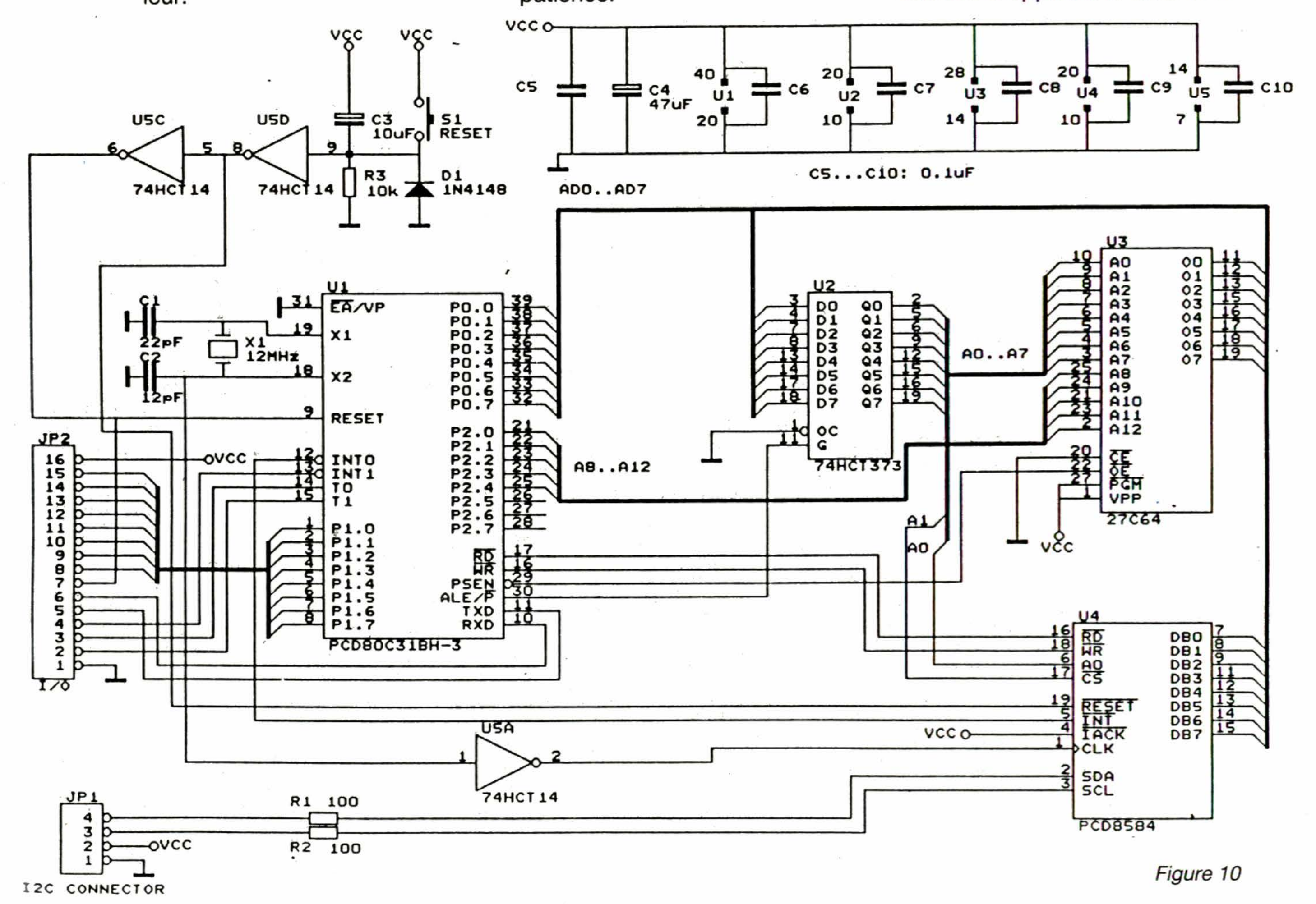


Figure 10

nés. Pas de commentaire technique particulier à ce sujet mais à noter qu'il est souvent préférable (pour des raisons bassement économiques) de passer aux 80C 652 ou 654 ou 552... qui comprennent l'interface "hard" I2C sur le cristal et que ce type de solution ne se justifie uniquement que lors de l'emploi complet (Timer 2 inclus) d'un 80C52 ou dans des solutions où des 80C51 sont déjà existants et immuables.
Vous trouverez un logiciel conçu pour cela dès le mois prochain sur le service minitel ERP.

Avec la famille 8088

Restons encore un peu dans l'architecture INTEL mais ici les signaux de service sont sensiblement différents et il est donc nécessaire de modifier les liaisons. Ceci est indiqué sur la **figure 11**.

Avec le Z80

Toujours dans la famille 8 bits et bien que son heure de gloire soit un peu dépassée, bon nombre d'entre vous l'utilise encore et ils n'ont pas le droit d'être privé d'I2C !
Nous les renvoyons avec plaisir à la **figure 12** et à leur minitel favori pour autres compléments.

Avec la famille 68000 et 90C xxx

Changement de décor et d'architecture. Cette famille à cœur de 68000 (16/32 bits) très performante a parfois besoin de s'occuper de nous les petits humains donc souvent d'utiliser les circuits I2C et souvent les interfaces hard ne sont pas à bord et... vive le PCD 8584 !

La **figure 13** vous donne ici aussi le schéma général et l'application concrète de ce genre d'applications.
Ici aussi des routines logicielles sont disponibles pour assurer l'interfaçage mais, afin de ne pas saturer ces pages, nous demandons à ceux qui seraient intéressés de se faire connaître sur le minitel ERP.

DES APPLICATIONS HORS DES SENTIERS BATTUS

Et elles sont nombreuses aussi...

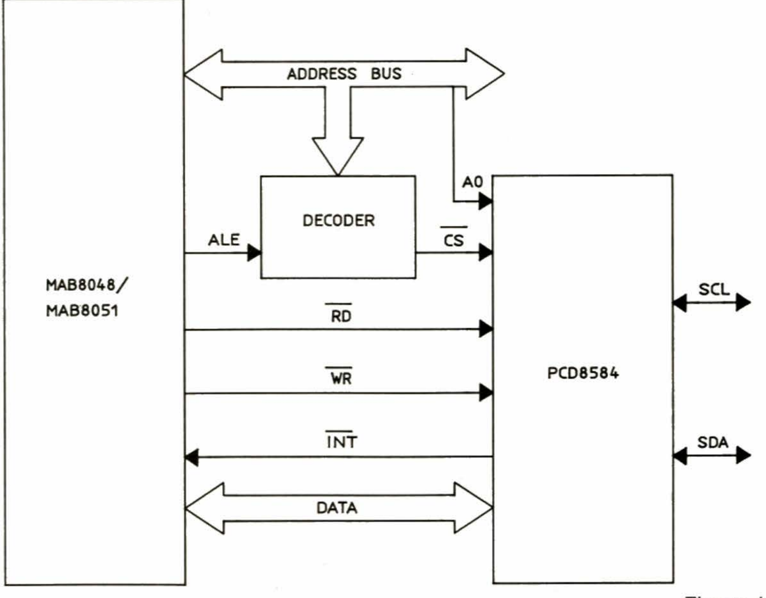


Figure 11

Lors de la conception, de nouvelles fonctionnalités ont été introduites, dites de "Spécial Modes".
- Un mode "moniteur".
- Un mode longue distance (ah !...).
- Un générateur de "strobe" (de fonction sonde).
Drôle de charabia... !

Le mode moniteur

Quand le registre S0' est chargé avec la valeur "00" (nous avons bien dit chargé — acte volontaire de votre part car il doit être mené en surimpression de la valeur de reset elle aussi égale à "00"), le PCD 8584 présente alors les particularités suivantes :
- Il est toujours sélectionné car

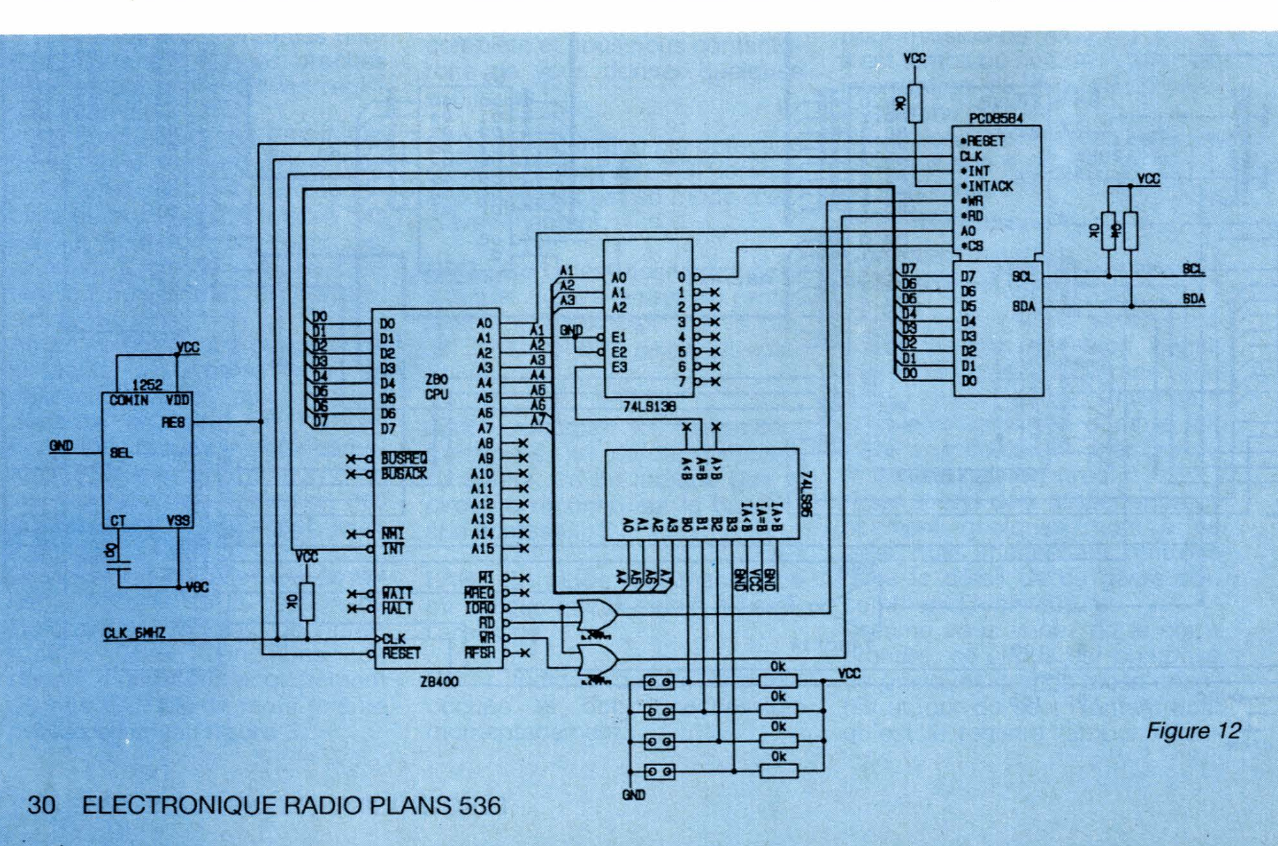


Figure 12

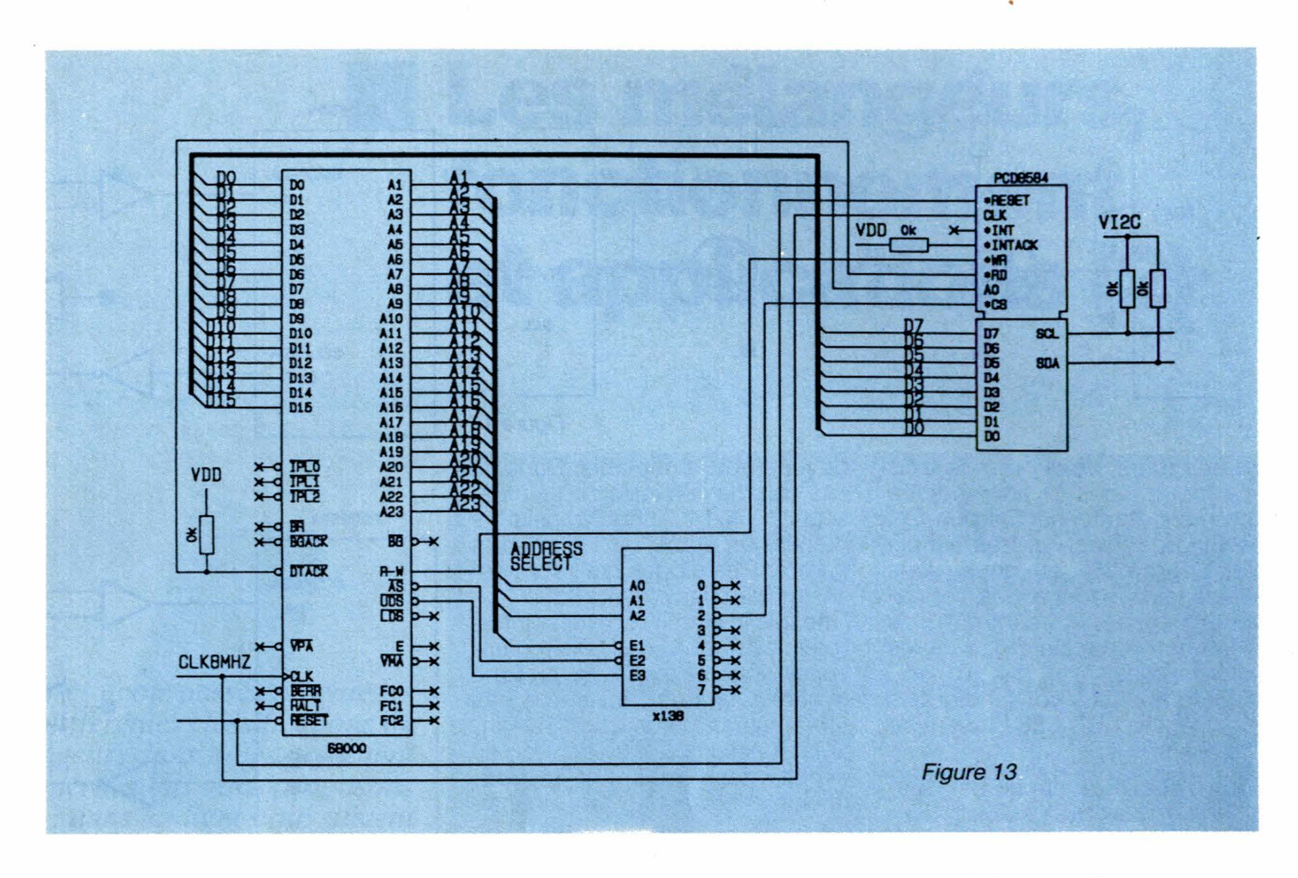


Figure 13

se mettant à l'écoute de tous (pour les spécialistes, c'est le "General Call" de l'I2C).

– Il passe automatiquement en position d'esclave-récepteur.

– Il fait le mort en ne renvoyant aucun acquittement à qui ce soit.

– Il fait la grève en ne générant aucune demande d'interruption.

– Les données qu'il reçoit sont directement aptes à être lues.

Résumons simplement : c'est un espion et c'est très laid mais bien pratique, surtout pour savoir ce qui se passe parfois sur le bus et développer des outils pour observer tous ses frasques et ses fantasmes (certains "disent" en bon français "le Monitorer). Notez au passage que depuis longtemps différentes Sociétés ont développé des produits commerciaux (Analyseurs et Aide au Développement de BUS I2C sur PC) selon ces principes.

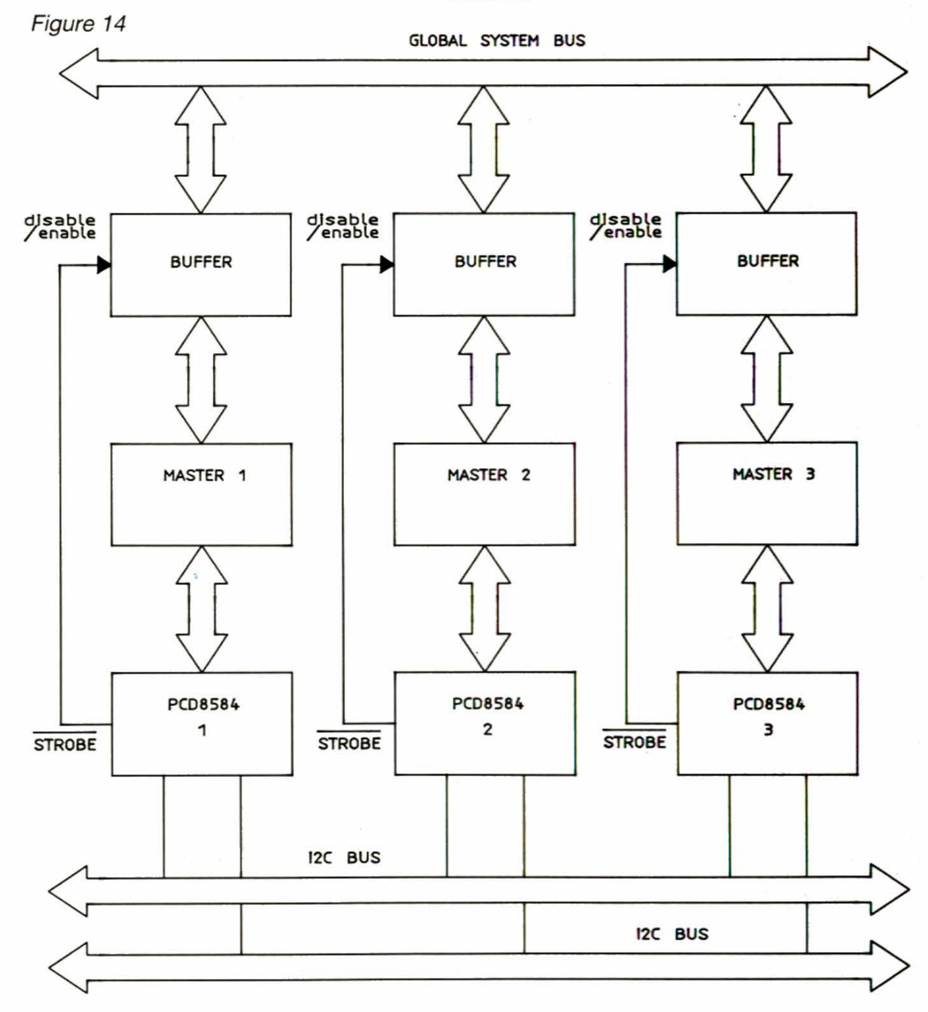
Le mode Strobe

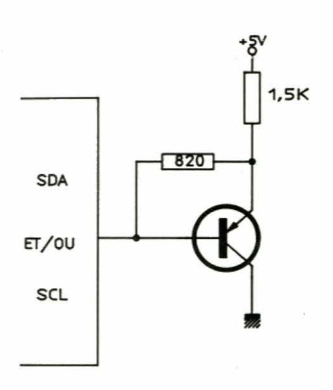
Esprit est-tu là ? Si tu est là, frappe un coup !
Et oui le PCD 8584 a été initié à des séances de spiritisme !
En effet lorsque le circuit reçoit un message comportant sa propre adresse, suivi immédiatement d'une conditiion de STOP, il comprend qu'il doit signaler sa présence en envoyant siné dié un signal dit de "STROBE" (sur sa broche de "RESET/STROBE", (active à l'état BAS) durant 8 périodes d'horloge CLK. Ça ouvre des horizons.
Un exemple d'architecture "d'horizon" est donné **figure 14**.

Le mode longue distance

Longue histoire... Peut-être un jour y consacrerons-nous tout un article.

Figure 14





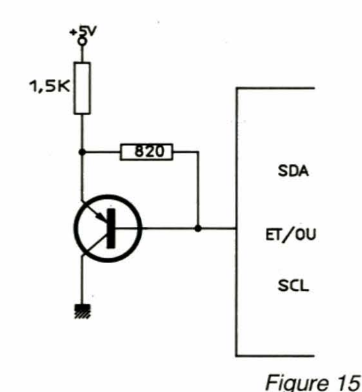


Figure 15

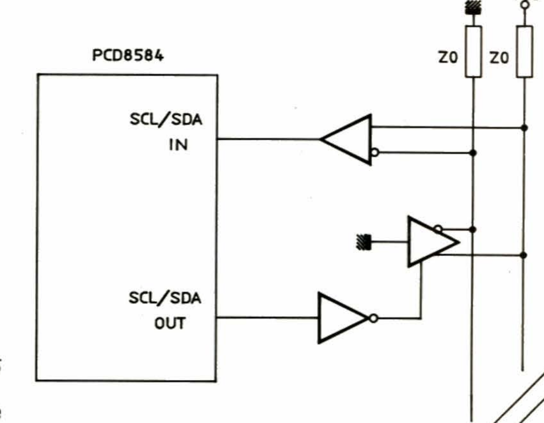


Figure 16

En attendant, voici près de trois ans nous vous avons indiqué comment "bufériser" le bus I2C de façon efficace et très simple (voir **figure 15**) et nous vous avions aussi donné des indications pour le transmettre de façon opto-couplée et sur paires différentielles.
Aujourd'hui nous vous indiquons comment le PCD 8584 résout ce problème.

Tout d'abord ce circuit est doté d'une part de "SDA out" et "SDA in" et d'autre part de "SCL out" et "SCL in" (voir **figure 2**), chacune d'entre elles étant UNIDIRECTIONNELLE.
Ces doubles entrées/sorties ont été conçues afin notamment d'être connectées à des circuits de commande de lignes longue distance (des "drivers" comme on dit), voir exemple n° 1.
D'autre part, de la même façon que les systèmes 4 – 20 mA, le PCD 8584 se veut de pouvoir gérer le bus I2C en "boucle de courant" or, pour qu'il y ait une boucle il faut qu'il y ait une entrée et une sortie. A ce sujet voir l'exemple n° 2.
Il est à noter que dans ces deux types d'architecture, la réception des données provenant des différents circuits disposés sur le réseau ne peut s'effectuer (simplement) qu'en mode de POLLING.
Deux exemples de principe sont donnés **figures 16** et **17**.

Exemple n° 1 :

Cet exemple illustre comment se servir des "in" et des "out" de même nom pour transformer le bus I2C de structure congénitale "asymétrique" en mode "symétrique" (type RS-485 ou RS-422A) à l'aide de circuits bien connus xxx 75176 ou xxx 96176 selon les marques.

Exemple n° 2 :

Ce deuxième exemple présente la possibilité de fonctionner en boucle de courant tout en étant de plus opto-couplable. Dans ce cas, lors de l'émission d'un message, les phototransistors d'émission étant "off", le courant parcourt les diodes qui informent les entrées des circuits I2C.
Une fois le circuit récepteur interrogé, celui-ci a tout loisir d'envoyer son message en mettant dynamiquement (au rythme des bits transmis) la broche "SDA in" à la masse, ce qui permet au circuit intégré maître de récupérer les informations tant souhaitées.
En cas de systèmes multi-maîtres, synchronisation et arbitrage du bus sont des choses périlleuses mais tout à fait réalisables.
Nous vous donnons rendez-vous le mois prochain pour la mise en œuvre matérielle et logicielle de ce compoant.
A bientôt donc....

Dominique PARET

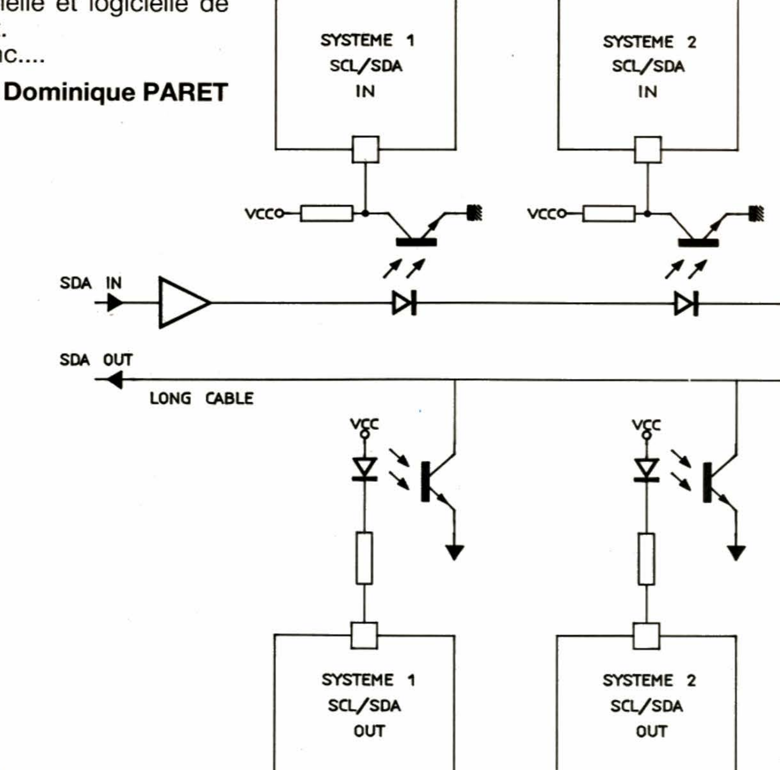
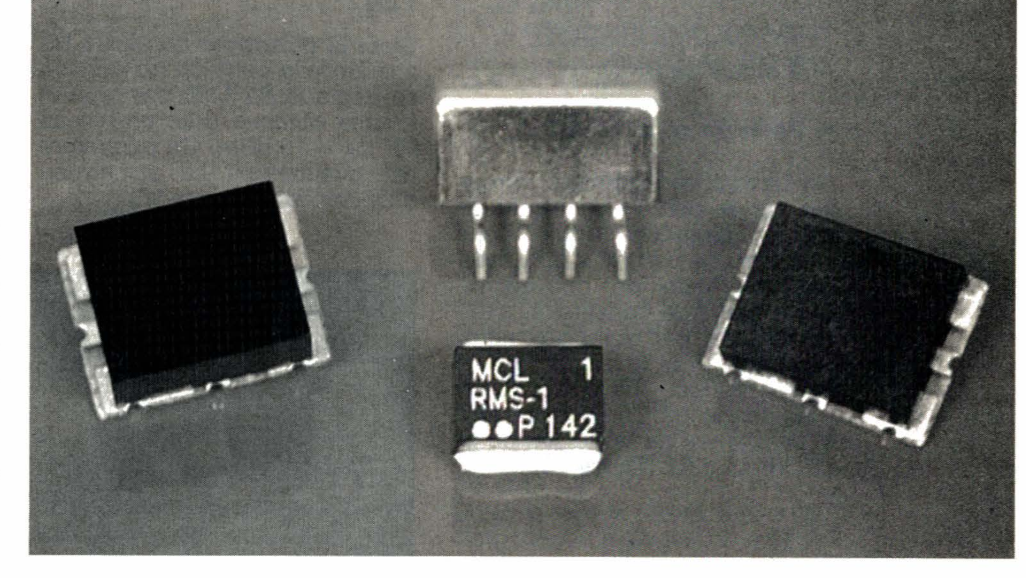


Figure 17

Les mélangeurs, fonctionnement et applications (2)

Après avoir passé en revue les différentes classifications de mélangeurs dans notre précédent numéro, nous vous proposons dans les lignes qui suivent d'examiner leurs principales applications en abordant les techniques optimales d'utilisation. Nous en profiterons pour aborder les critères de sélection selon les applications envisagés.



TRANSPOSITION DE FRÉQUENCE :

L'utilisation la plus courante des mélangeurs en anneau est très certainement la transposition de féquence. Cette opération consiste à translater le spectre du signal RF autour de la fréquence intermédiaire FI et s'obtient par le mélange d'un oscillateur local fort niveau, appliqué sur l'OL, avec le signal RF. Il en résulte la création des fréquences somme et différence $f_{OL} + f_{RF}$ et $f_{OL} - f_{RF}$ (**figure 17**).

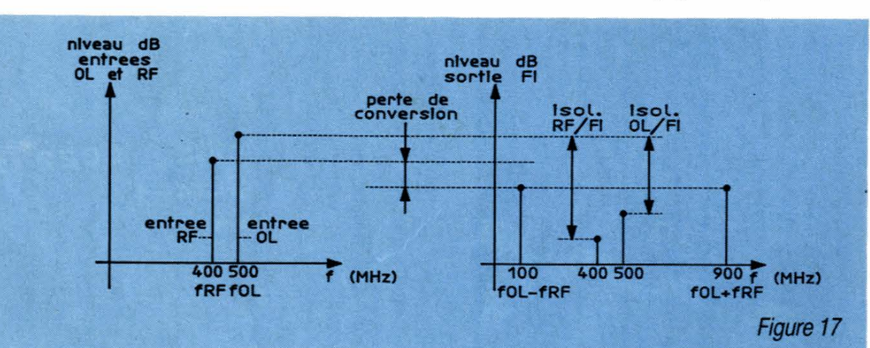


Figure 17

La translation peut se faire vers les hautes fréquences, et ceci sera le cas lorsque le traitement à effectuer sur le signal est plus facile à réaliser en basse fréquence. Le passage en haute fréquence se faisant alors au niveau du dernier étage pour un émetteur. Il faut filtrer le spectre $f_{OL} - f_{RF}$ si celui-ci est gênant pour l'application qui se situe, elle, à $f_{OL} + f_{RF}$.

La translation peut se faire vers les basses fréquences, et c'est le cas de tous les récepteurs MF, AM, CB, etc. qui ramènent le signal à 10,7 MHz (FI standard). Le filtrage du spectre supérieur par un passe-bas ou passe-bande étant simple à réaliser.

Les deux bandes latérales $f_{OL} - f_{RF}$ et $f_{OL} + f_{RF}$ ont en général même amplitude. L'amplitude est fonction de la réponse en fréquence du mélangeur. Ce dernier ne donnera une égalité d'amplitude que dans sa plage d'utilisation.

Les ports des mélangeurs peuvent être interchangeables et en particulier les ports FI et RF. De manière générale la bande passante du port FI va du DC à quelques MHz (par exemple), alors que celle du port RF ne commencera qu'à 10 MHz pour

couvrir jusqu'à quelques 100 MHz (par exemple). Dans le cas d'une translation d'un signal de 3 MHz vers 13 MHz, le 3 MHz peut être injecté sur FI et le 10 MHz sur l'OL. On obtiendra alors sur le port RF les signaux à 7 MHz et à 13 MHz, compatible avec les bandes passantes des différents ports.
L'oscillateur local est toujours injecté sur l'OL car ce port présente le plus fort isolement et le fort niveau d'OL à appliquer sera moins perturbant en sortie.

TERMINOLOGIE ASSOCIÉE AUX MÉLANGEURS :

• Perte de conversion :

On devrait normalement retrouver sur le port FI un signal de même amplitude que celui injecté sur le port RF s'il n'y avait pas de perte de conversion. Cette perte de conversion s'exprime donc par le rapport entre le signal RF et le signal FI à la fréquence $f_{OL} - f_{RF}$ ou $f_{OL} + f_{RF}$. Ces pertes de conversion sont dépendantes de la fréquence et du niveau d'OL (**figure 18**).

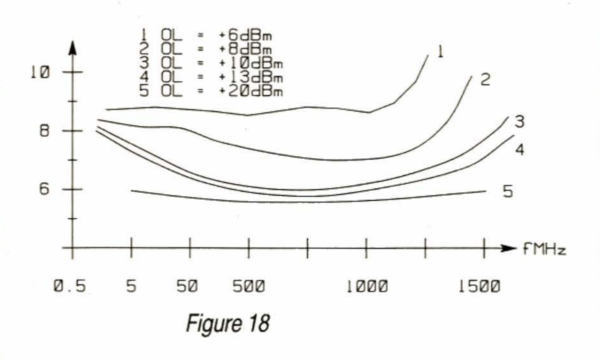


Figure 18

• Figure de bruit :

C'est le rapport entre le rapport signal à bruit à l'entrée FI et le rapport signal à bruit de $f_{OL} - f_{RF}$ (ou $f_{OL} + f_{RF}$) à la sortie RF (**figure 19**)

$$NF(dB) = 10 \log \left(\frac{S/N\ FI}{S/N\ RF} \right)$$

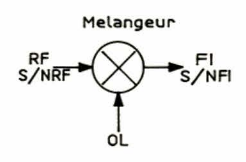


Figure 19

• Isolement :

Les fréquences f_{OL} et f_{RF} sont indésirables en sortie du mélangeur. L'isolement est l'atténuation que subit f_{OL} (ou f_{RF}) entre l'entrée OL (ou RF) et la sortie FI. Du fait des niveaux forts appliqués à l'OL, l'isolement doit être très fort afin d'éviter les perturbations la raie de l'OL en sortie.

• Point de compression à 1 dB :

C'est la puissance du signal RF à laquelle la perte de conversion se dégrade de 1 dB par rapport à celle obtenue pour un niveau RF de − 15 dBm. Ce paramètre est directement lié au niveau d'OL appliqué (**figure 20**).

• Niveau de sensibilisation à 1 dB :

C'est le niveau d'une raie parasite présente dans le signal RF au côté de la raie de signal utile d'amplitude − 15 dBm qui provoquera une dégradation de 1 dB de la perte de conversion. Ce niveau est inférieur ou égal au point de compression à 1 dB. Le mélangeur sera d'autant meilleur que le niveau de désensibilisation à 1 dB se rapprochera du point de compression à 1 dB.

• Produit d'intermodulation :

Les non-linéarités du mélangeur provoque l'apparition de fréquences parasites qui devraient être absentes s'il était parfait. Les fréquences désirées sont $f_{OL} - f_{RF}$ et $f_{RF} + f_{OL}$.

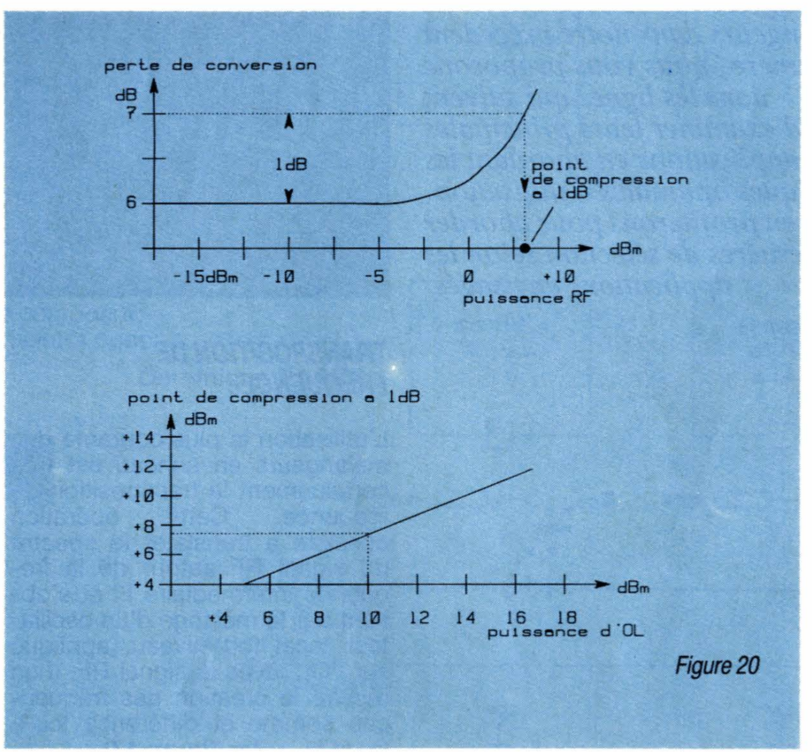
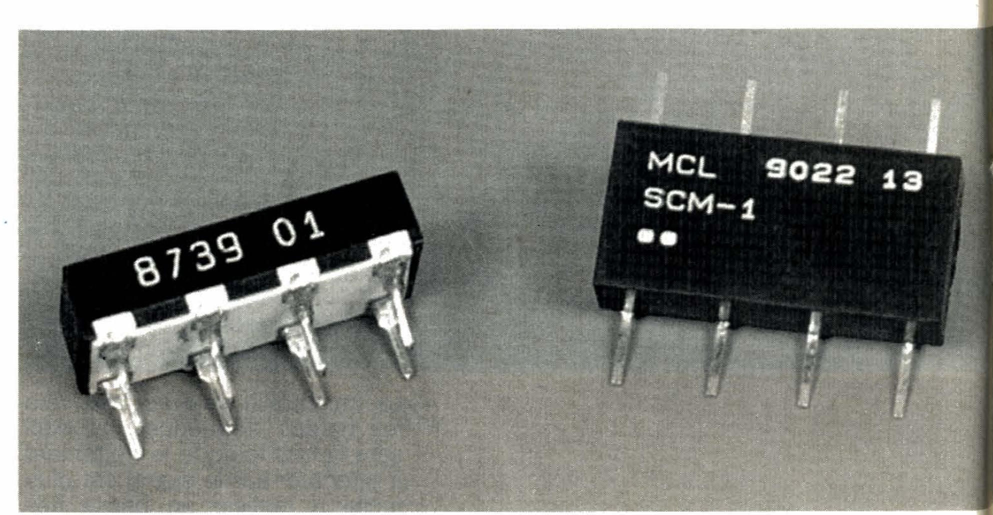


Figure 20



Les fréquences indésirées sont $f_{OL} + 2f_{RF}$, $f_{OL} - 2f_{RF}$, $2f_{OL} + f_{RF}$, $2f_{OL} - f_{RF}$, etc. soit $Nf_{OL} \pm Mf_{RF}$. Ces produits d'intermodulation prennent naissance en quasi totalité dans le seul élément non linéaire du mélangeur, c'est-à-dire les diodes constituant l'anneau de commutation. C'est pour cela qu'il existe différentes classes de mélangeur. Un exemple comparatif de réjection du produit d'intermodulation $Nf_{RF} \pm Mf_{OL}$ est représenté aux **figures 21 a** à **d**. Il est à remarquer que la classe IV est de loin la plus performante. Par contre, les classes II ne sont pas toujours meilleures que la classe I. Il faudra donc bien analyser le besoin en performances et ne considérer que les fréquences parasites gênantes dans le spectre utile car ne pouvant pas être supprimées par filtrage.
Ces divers termes techniques permettent de mieux comprendre les data sheets et également d'être conscient des imperfections des mélangeurs en anneau. Dans tous les cas il faut tout d'abord bien poser le problème et maîtriser l'environnement système dans lequel le mélangeur doit fonctionner avant de le choisir. En effet, c'est à partir des donnés d'environnement qu'il faut parcourir les data books à la recherche du composant adéquat.

Influence de l'environnement sur le mélangeur :

Afin d'obtenir des performances reproductibles, et plus particulièrement lors de l'utilisation de mélangeurs classiques, le concepteur doit connaitre les effets des réflexions dues aux désadaptations vues par le mélangeur. Toutes les désaptations de source et de charge peuvent affecter les performances.



Mais l'effet sera plus ou moins important selon le port concerné. Le tableau 1 donne les effets relatifs d'une désadaptation large bande de la source ou de la charge vis-à-vis des paramètres principaux des mélangeurs classiques (classes I à III) et du mélangeur de classe IV (insensible à la nature de la charge).
La désadaptation des ports RF et FI a le plus d'importance dans les effets néfastes sur les performances.

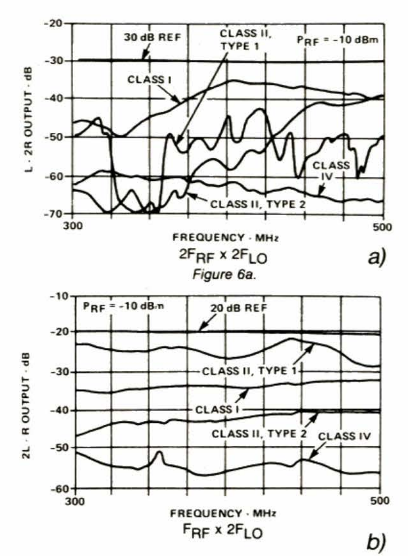


Figure 6a.

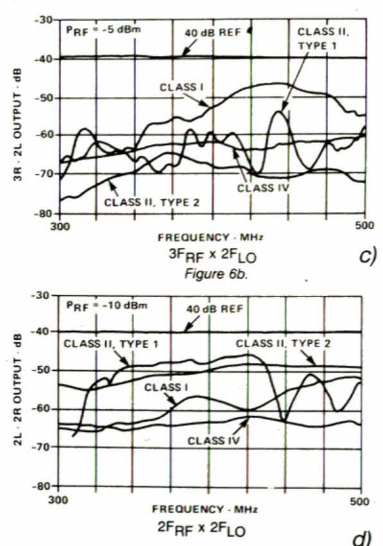


Figure 6b.

Figure 21

INTERFAÇAGE DES MÉLANGEURS :

Le tableau susmentionné met en évidence que:
– Les mélangeurs classiques (classes I à III) sont très sensibles à la désadaptation du port FI. Dans la plupart des applications une seule des fréquences ($f_{OL} - f_{RF}$) et ($f_{OL} + f_{RF}$) est utile. Mais la plupart des mélangeurs sortiront les deux avec des amplitudes égales et plus particulièrement lorsqu'on est intéressé par $f_{OL} + f_{RF}$. De plus, tous les mélangeurs possèdent un produit d'intermodulation $3f_{OL} \pm f_{RF}$ important (environ – 13 dBc "voir annexe 2"). Le spectre sur le port FI sera donc constitué des raies $f_{OL} - f_{RF}$, $f_{OL} + f_{RF}$, $3f_{OL} - f_{RF}$ et $3f_{OL} + f_{RF}$. Le fait de filtrer en sortie du mélangeur, afin de récupérer la raie unique $f_{OL} + f_{RF}$ (ou $f_{OL} - f_{RF}$) nous intéressant, provoque une désadaptation de la charge en dehors de la bande de travail. En effet, l'opération réalisée dans un filtre passe-bas par exemple consiste à présenter une impédance de transfert nulle dans la bande passante et une impédance de transfert infinie en dehors.

Dans le circuit de la **figure 22**, l'impédance Z sera voisine de R_0 pour $f \leqslant f_c$, égale à $R_0/2$ à $f = f_c$ et voisine de 0 pour $f \geqslant f_c$. Le port FI du mélangeur sera donc com-

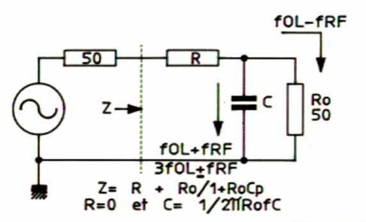


Figure 22

Influence sur les paramètres du mélangeur	désadaptation de la source OL		désadaptation de la source RF		désadaptation de la charge FI	
classe	I à III	IV	I à III	IV	I à III	IV
Perte de conversion	F	F	I	F	M	F
Produit d'intermodulation	F	F	M	F	I	M
Distorsion par interm. du 3e ordre	F	F	M	F	I	F
Cœff. de réflexion du port RF	F	F	NA	NA	I	I
Cœff. de réflexion du port Fi	F	F	I	I	NA	NA

F = faible M = Moyenne I = Importante NA = Non Applicable

Tableau 1

plètement désadapté et d'après le tableau ci-dessus provoquera une forte dégradation des performances souhaitées. Ce n'est pas le mélangeur qu'il faut mettre en cause, mais le circuit l'environnant. Il faut donc trouver une solution telle que sur la charge souhaitée on obtienne une bande passante f_c et que le mélangeur voie sur son port FI une impédance constante sur sa bande passante propre. La réalisation d'un passe-bas avec un condensateur à la masse ne permet pas de résoudre le problème car elle représente un court-circuit en HF et la charge vue de FI sera toujours nulle en HF quoi qu'il arrive. Il est nécessaire de réaliser le filtre passe-bas avec une self série qui présentera une impédance infinie en HF et coupera donc le signal comme souhaité. Dans ce cas, il est possible de mettre en parallèle sur la charge utile un circuit qui chargera le port FI pour les fréquences supérieures à f_c (**figure 23**).

– Quand f très grande, donc p très grand, alors $Z \simeq R_0$.

– Quand f = 0, donc p = 0, alors $Z = R_0$.

– Quand $f = \dfrac{1}{2\pi\sqrt{LC}}$,

$$\text{Alors } Z = R_0 \frac{2 + R_0\sqrt{\dfrac{C}{L}} + \dfrac{1}{R_0}\sqrt{\dfrac{L}{C}}}{2 + 2R_0\sqrt{\dfrac{C}{L}}}$$

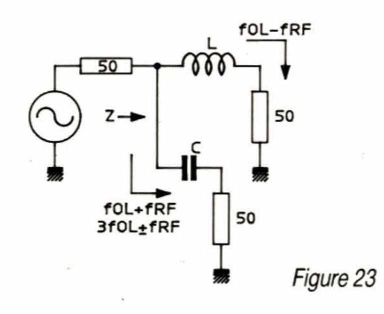


Figure 23

$$Z = \frac{1}{Y} \text{ et } Y = \frac{1}{Lp + R_0} + \frac{1}{R_0 + \dfrac{1}{Cp}}$$

$$= \frac{1 + 2p\,R_0C + LCp^2}{R_0 \cdot [1 + (R_0C + \dfrac{L}{R_0})p + LCp^2]}$$

$$Z = R_0 \frac{1 + (R_0C + \dfrac{L}{R_0})p + LCp^2}{1 + 2\,R_0Cp + LCp^2}$$

donc en prenant :

$$R_0\sqrt{\frac{C}{L}} = \frac{1}{R_0}\sqrt{\frac{L}{C}}$$

$$\text{soit } R_0^2 = \frac{L}{C}$$

on obtient $Z = R_0$ et les impédances des deux branches sont identiques donc

$$f_c = \frac{1}{2\pi\sqrt{LC}}$$

Le port FI verra toujours l'impédance R_0 et les paramètres du mélangeur resteront corrects.

Le circuit à adopter pour le filtre passe-haut est le même. La charge utile étant celle de la branche capacitive.

Le circuit à adopter pour un filtrage du 3e ordre passe-bas sera celui de la **figure 24**.



de produit d'intermodulation $Nf_{OL} \pm M f_{RF}$, de distorsion par intermodulation du 3e ordre et les pertes de conversion. L'utilisation de filtres complémentaires tels que ceux décrits plus haut et communément appelés "Diplexer" permet de s'affranchir de ce genre de problème. Lorsqu'il est nécessaire de travailler avec la fréquence $f_{OL} + f_{RF}$, il faut utiliser un filtre passe-bande dont le schéma de la **figure 25** montre une réalidation possible.

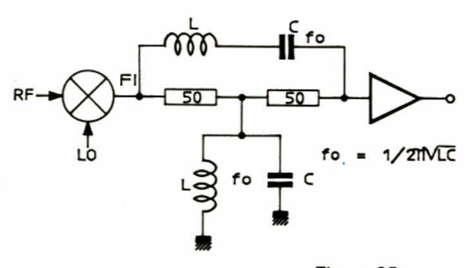
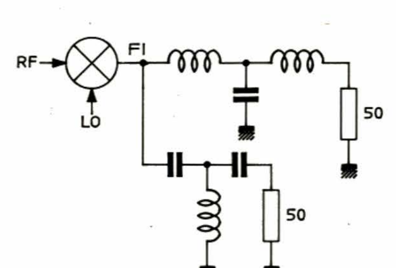


Figure 25



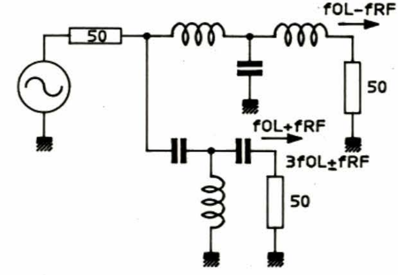
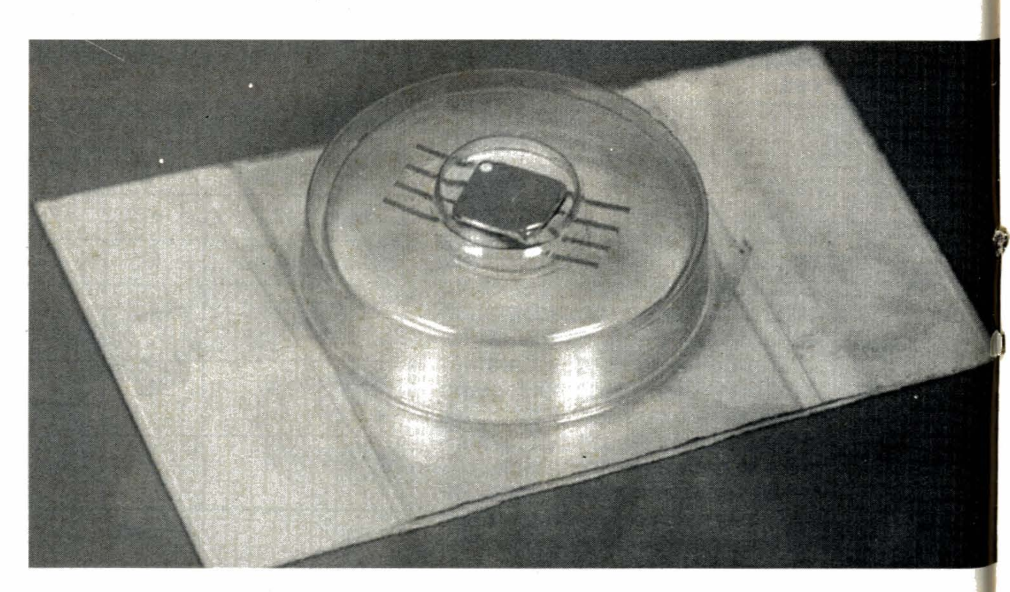


Figure 24

Lorsque le port FI est désadapté, le signal réfléchi retourne dans le mélangeur par le port FI et se retrouve mélangé avec l'oscillateur local. Cette situation affecte très fortement les performances

Seule la fréquence désirée traverse le circuit LC série et les signaux de fréquence inférieure et supérieure sont bloqués et ne peuvent "s'écouler" que par les résistances de 50 Ω. Le circuit



LC parallèle présente une forte impédance à la fréquence de résonance f_0 et devient un "court-circuit" en delà et en deçà. Ceci conduit à charger le mélangeur FI par 50 Ω connectée à la masse à travers le circuit LC parallèle pour les fréquences $f < f_0$ et $f > f_0$. La structure du circuit permet également de charger l'étage d'entrée de l'ampli qui suit avec une charge constante en fonction de la fréquence. Le circuit est totalement symétrique. Les filtres à très faible bande passante sont très difficiles à réaliser du fait des Q élevés nécessaires et de la complémentarité parfaite nécessaire pour les deux circuits LC série et LC parallèle.
Une autre technique de filtrage beaucoup plus complexe mais dont les performances méritent un développempent est représentée sur le schéma de la **figure 26**. Ce filtre est particulièrement utilisé lorsque seule la fréquence $f_{OL} + f_{RF}$ est intéressante.

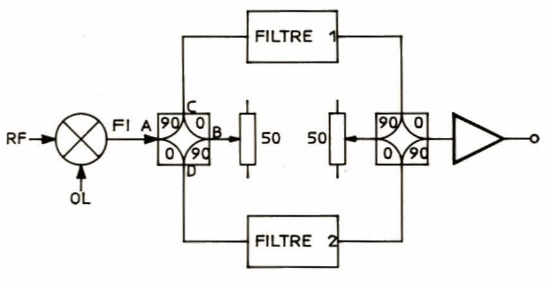
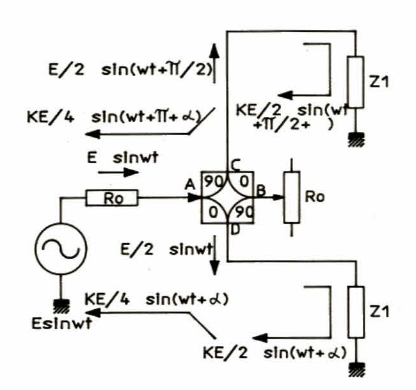


Figure 26

Les deux diviseurs de puissance "Hybrid devices" ont la particularité de produire 2 voies d'égales amplitudes mais déphasées de 90°. Le port opposé présente une isolation totale. A et B (C et D) sont donc isolés.
Ils peuvent fonctionner comme des diviseurs de puissance, dans ce cas un seul port est en présence d'une source de signal, comme c'est le cas dans notre application, le port FI du mélangeur attaquant le port A de l'"hybrid". Ils peuvent également fonctionner en additionneur de puissance lorsqu'ils sont en présence de 2 sources attaquant des ports opposés (A et B) ou (C et D). Les ports A-C, A-D, C-B, B-D sont bidirectionnels et présente les mêmes caractéristiques de phase. C'est-à-dire que lorsqu'un signal est appliqué sur le port A il sera déphasé de 90° sur le port C. Le même signal appliqué au port C sera également déphasé de 90° à la sortie du port A.
Dans notre application, une désadaptation du filtre 1 provoquera une réflexion et le signal ainsi réinjecté sur le port C sera déphasé de 90° sur son chemin retour.
Ayant subi le même déphasage sur son chemin aller, le déphasage total sera de 180° donc en opposition de phase avec le même signal injecté en A, réfléchi du fait de la désadaptation du filtre 2 et de retour en A avec un déphasage nul dans l'"hybrid". Si les filtres 1 et 2 sont identiques (**figure 27**), les réflexions dues



$Z_1 \neq R_0$
d'où le retour en a de

$$k\frac{E}{4}\sin(\omega t + \pi + \propto) + k\frac{E}{4}\sin(\omega t + \propto)$$

$$= -k\frac{E}{4}\sin(\omega t + \propto) + k\frac{E}{4}\sin(\omega t + \propto)$$

$$= 0$$

Figure 27

aux désadaptations s'annuleront par recombinaison sur le port A. La charge placée sur le port B permet de ne pas réinjecter de signaux par désadaptation de l'"hybrid".

L'"hybrid" placé en sortie de notre application permet de récupérer la puissance totale par sommation des signaux qui avait été divisés en 2 à l'entrée. Les déphasages de l'hybrid de sortie sont placés à l'opposé de l'"hybrid" d'entrée afin de retrouver le même état de phase des deux branches et d'opérer la sommation en sortie. Le comportement de l'"hybrid" de sortie vis-à-vis du circuit qu'il attaque est exactement le même que celui d'entrée vis-à-vis du mélangeur. Les charges vues de l'entrée et de la sortie sont parfaitement adaptées.

Il est possible d'obtenir des filtrages à bande très étroite avec ce type de schéma.

QUELQUES APPLICATIONS DE MÉLANGEUR :

Détecteur de phase :

L'application de deux signaux sinusoïdaux de même fréquence sur les ports RF et OL du mélangeur conduit à retrouver en sortie le signal FI tel que :
$RF = A_1 \cos(\omega_1 t + \varphi_R)$
$OL = A_2 \cos(\omega_1 t + \varphi_0)$

$$Fi = RF \times OL$$
$$= \frac{A_1 A_2}{2}[\cos(\omega_1 t + \varphi_R + \omega_1 t + \varphi_0) + \cos(\omega_1 t + \varphi_R - \omega_1 t - \varphi_0)]$$
$$= \frac{A_1 A_2}{2}[\cos(\varphi_R - \varphi_0) + \cos(2\omega_1 t + \varphi_R + \varphi_0)]$$

La raie à $2f_1$ filtrée par un passe-bas conduit à ne retrouver en FI qu'un signal continu représentatif de l'écart de phase des signaux RF et LO (**figure 28**).

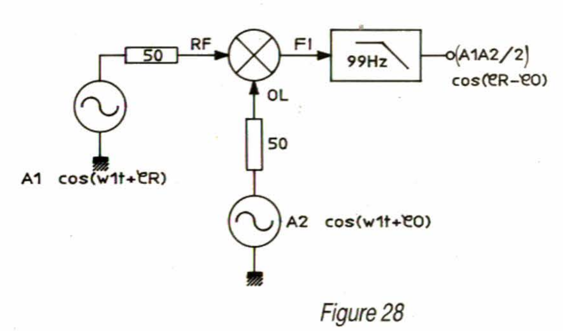


Figure 28

Avec des signaux RF et OL sinusoïdaux, le signal en sortie est sinusoïdal (**figure 29**) et présen-

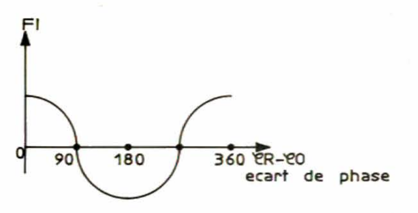


Figure 29

tera une tension nulle pour un déphasage relatif de 90°. L'utilisation d'un tel détecteur dans un démodulateur (type FM par exemple) asservira l'oscillateur local sur un écart de phase de 90° car autour de cette valeur la courbe ne présente pas de changement de signe de sa pente.
L'asservissement est convergent et stable sur le point à 90°.



L'application des signaux carrés conduit à une courbe linéaire en sortie FI après élimination par filtrage de la composante à 2 fois la fréquence (**figure 30**).

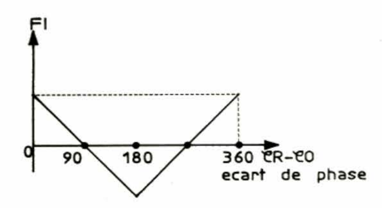


Figure 30

Le grand problème avec ce type de détecteur de phase réside dans l'erreur d'offset due aux imperfections du mélangeur. Le mauvais appariement des diodes constituant le pont en anneau et la dissymétrie des transofrmateurs OL et RF sont les deux causes principales de ce problème. Il est possible de compenser cet offset en appliquant sur le port FI un courant continu de manière à dissymétriser la polarisation des diodes de l'anneau. Le réglage se faisant en appliquant le même signal sur OL et RF déphasé exactement de 90° (**figure 31**).

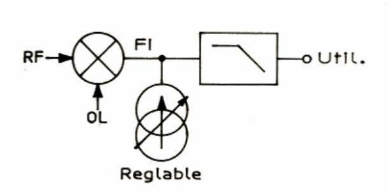


Figure 31

Les mélangeurs en modulateur :

Dans tous les types de modulation : phase, amplitude, impulsion, etc., la porteuse est appliquée sur le port OL, le signal modulant sur le port FI et le signal modulé se retrouve sur le port RF (**figure 32**).

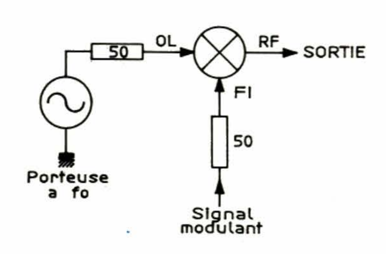


Figure 32

a) Modulation d'amplitude et atténuateur contrôlé en courant :

La proportion de signal OL traversant le mélangeur pour ressortir en RF est fonction du courant continu appliqué sur le port FI (**figure 33**). Lorsque ce cou-

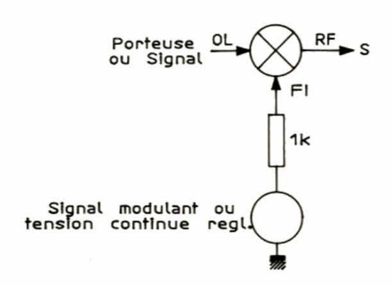


Figure 33

rant est nul, on ne retrouvera rien en sortie RF. L'atténuation est maximale. L'atténuation minimale sera obtenue pour des courants de l'ordre de 20 mA. En appliquant un courant moyen de 1 mA, par exemple, on se placera loin des non-linéarités de la courbe de transfert (**figure 34**).

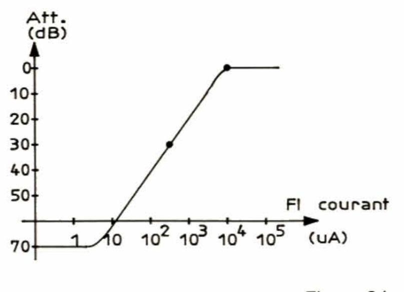


Figure 34

La superposition d'un courant de modulation provoquera une modulation d'atténuation et donc une modulation d'amplitude de la porteuse OL. Il est nécessaire de connaitre la courbe d'atténuation afin de travailler dans la bonne région du mélangeur conduisant à la distorsion minimale.

b) Modulation biphase :

Les deux états de phase 0° et 180° sont obtenus en appliquant un courant entrant et sortant du port FI.
En injectant un courant entrant dans le port FI, il forcera les diodes D_1 et D_4 à conduire. Les diodes D_2 et D_3 resteront bloquées.
le schéma ainsi obtenu (**figure 35**) du point de vue HF est donc la connexion des transfos OL et RF de telle sorte que le signal OL se retrouve en RF avec un déphasage nulle. L'injection d'un courant sortant du port FI forcera les diodes D_2 et D_3 à conduire. Cette fois il y a inversion de phase de 180° des signaux OL et RF.
Ce type de modulation est très employée dans les modems.

c) Modulation quadriphase :

Afin d'obtenir 4 états de phase avec des mélangeurs, il faut utiliser deux modulateurs biphase. Le schéma d'un tel modulateur est plus complexe que le précédant (**figure 36**).

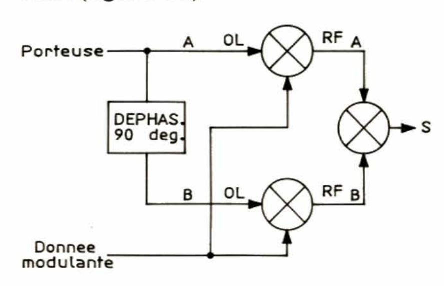
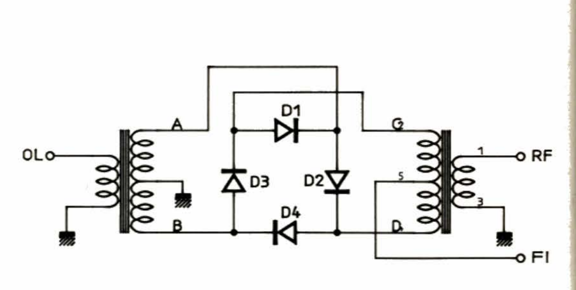
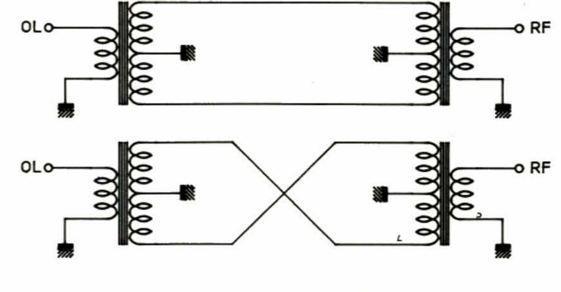


Figure 36





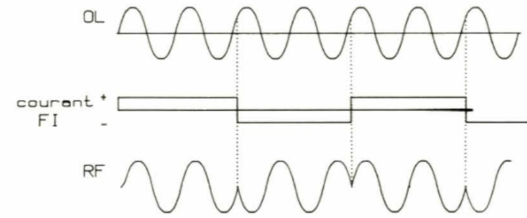


Figure 35

Les signaux RFA et RFB sont additionnés pour former la sortie.

J.-Y. Bedu

Les principaux fabricants de mélangeurs :

ANZAC, WATKINS-JOHNSON, MINI-CIRCUITS, EUROTEC SYNERGY.

ANNEXE 1

Multiplication de deux signaux sinusoïdaux soit $v_1 = A_1 \cos \omega_1 t$ et $v_2 = A_2 \cos \omega_2 t$, le produit $v_1 \times v_2 = P$ aura pour expression :

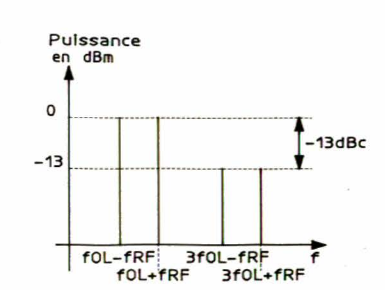
$$P = v_1 \times v_2 = A_1 A_2 \cos \omega_1 t \cos \omega_2 t$$

$$= \frac{A_1 A_2}{2} [\cos(\omega_1 + \omega_2)t + \cos(\omega_1 - \omega_2)t]$$

ANNEXE 2

Définition du dBc (dB cycle).
Cette grandeur étant exprimée en dB représente donc un rapport et exprime une quantité relativement à une autre. L'indice c signifie "carrier" en anglais c'est-à-dire "porteuse" en français. Dans notre cas précis il s'agit de la raie de fréquence $f_{OL} + f_{RF}$ ou $f_{OL} - f_{RF}$.
La puissance du produit d'intermodulation qui nous intéresse sera donc :

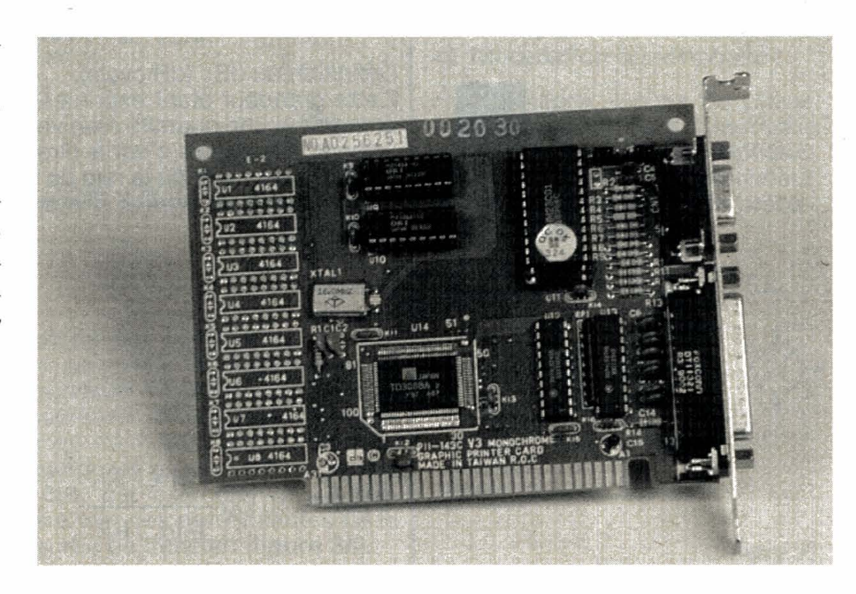
$$P_{(dBc)} = 10 \log \frac{P_{(3f_{OL} \pm f_{RF})}}{P_{(f_{OL} + f_{RF})}}$$



Si l'amplitude de $f_{OL} \pm f_{RF}$ était de de + 4 dBm alors – 13 dBc conduirait à une amplitude de – 9 dBm pour les raies $3 f_{OL} \pm f_{RF}$

Interface Parallèle Bidirectionnelle pour PC

L'interface imprimante du PC est unidirectionnelle, les données vont du PC vers l'imprimante mais certaines applications demandent une configuration en entrée ; nous allons voir qu'il est possible de modifier cette interface pour un fonctionnement en mode bidirectionnel, si ce n'est pas déjà le cas !



Un peu d'histoire

L'interface imprimante pour le PC utilise le protocole Centronics, du nom du fabriquant qui a mis sur le marché ce système de transmission de données en mode parallèle.
Conçue par un fabricant d'imprimantes, cette interface a été prévue pour fonctionner en mode unidirectionnelle, les données vont du PC vers l'imprimante.
Toutes les interfaces PC (d'IBM) XT, AT ou de clones "hors CEE" ont repris le même schéma, ils utilisent toujours les mêmes circuits, quelquefois les numéros (Uxx) et le cablage électrique sont identiques à la version IBM d'origine.
Certains fabriquants français se sont distingués en ne reprenant pas le schéma d'origine (utilisation de PAL), la modification de ces cartes suivant le descriptif qui suit ne pourra pas être fait, une analyse plus détaillée de la carte devra être faite.
La souplesse d'utilisation de cette interface et sa rapidité l'ont fait évoluer, IBM a équipé son PS d'une interface bidirectionnelle.
Certains portables en sont équipés, nous allons voir comment équiper son PC d'une interface de ce type.

LES CONFIGURATIONS

Etudions les différents types de PC et comment arriver à installer cette interface bidirectionnelle.
Sur un vieux PC, de première génération (XT), équipé d'une carte imprimante (cuivre double face et circuits classiques) couplée à une carte vidéo MDA (monochrome) ou CGA (même sur une carte mémoire d'extension), une légère modification hardware décrite ci-après suffit.
Sur un PC très moderne (équipé de gros VLSI plein de pattes), cette interface bidirectionnelle existe peut-être (?), pour le savoir il suffit de lancer le programme BASIC inclus dans cet article. Ce logiciel détecte les interfaces "bidirectionnelles".
Si c'est un succès, l'interface est présente, dans le cas contraire, avant de passer à la solution de secours, des essais pourront être entrepris en plaçant ou en enlevant des cavaliers de configurations "inconnus" souvent présents sur ces types de cartes.
Pour le moment aucune modification hardware n'est prévue, simplement parce que ces cartes sont difficilement modifiables et surtout par manque d'informations sur ces boîtes noires.

La solution de secours

Si l'on possède une carte vidéo/imprimante VLSI coûteuse, ou si le logiciel et les différents essais de configuration avec les cavaliers n'ont rien donné, on doit se rabattre sur cette solution.
Dans l'espace du PC, deux zones sont prévues pour les imprimantes.
Il suffit d'installer une vieille carte imprimante dans l'espace libre.
Avec ce jeu de cartes (!), on en revient à la solution de la modification d'une vieille carte PC.

Modification d'une carte

Il est primordial que cette modification laisse l'interface imprimante compatible ; ainsi à la mise sous tension et pour une utilisation normale en sortie imprimante, aucune différence n'apparaît avec le fonctionnement d'origine.
Le schéma de la **figure 1** donne un plan simplifié de l'interface imprimante.
En observant les circuits autour de l'interface imprimante, on peut savoir rapidement si une carte sera modifiable ou non.
La présence du 74LS174, du 74LS374, et du 74LS244, 74LS245 laisse à penser que cette interface est modifiable.
Le bus du PC est "bufferisé" par un 74LS245 (non représenté), ce bus interne à la carte va permettre la lecture ou l'écriture sur le port DATA, sur le port CONTRÔLE, et en lecture seule, le status de l'imprimante.

Trois adresses dans l'espace I/O du PC, dont deux en lecture/écriture, constituent l'espace nécessaire pour une interface parallèle. Deux zones sont prévues dans le PC.
Pour information, la **figure 3** donne la correspondance entre les bits des registres et la prise cannon 25 points.
Sur la **figure 1**, du côté DATA, un 74LS374 présente les données sur le port imprimante, un 74LS244 permet de relire les données.
IBM précise que cette lecture permet de vérifier l'état de sortie du port imprimante, en vue d'une détection et d'un contrôle de l'interface.
La patte OE du 74LS374 étant à la masse, les données sortent tout le temps, il n'est pas possible dans ces conditions d'utiliser le port en entrée.
Pour passer le port en configuration entrée, nous allons couper cette liaison afin de pouvoir mettre à 1 l'entrée OE du 74LS374.
En analysant la partie CONTRÔLE de la carte, on s'aperçoit que les six fils du bus de données sont reliés sur le 74LS174 qui gère les signaux de contrôle (et qui contient six registres).
Hors seulement 5 des six signaux disponibles sont utilisés. Le sixième est non connecté (NC), il suffit de le relier à l'OE du 74LS374 et la modification est terminée (**figure 2**) (mais c'est bien sûr !).
Cette modification est si évidente qu'on peut se demander pourquoi elle n'a pas été faite dès le début, à moins que...
A la mise sous tension, une remise à zéro est effectuée sur le 74LS174, la patte OE est mise à zéro, c'est la configuration standard, le BIOS du PC relie par le 74LS244 les informations sortantes du 74LS374, la détection de la carte imprimante s'effectue correctement.
Comme nous l'avons vu, seuls cinq des huit fils que contient le bus de données sont utilisés pour le contrôle. Les logiciels de contrôle de l'imprimante positionnent les trois derniers bits à zéro, donc reconfigurent systématiquement le port dans sa configuration imprimante.
Pour cette modification nous avons utilisé le bit 5, certains constructeurs l'utilisent aussi, d'autres le 6 ou le 7 ; pour le savoir un petit programme en BASIC va chercher la configuration, il permettra aussi de vérifier le cablage correct de la modification matérielle.
Un petit montage à LED, en option, permet de visualiser l'état

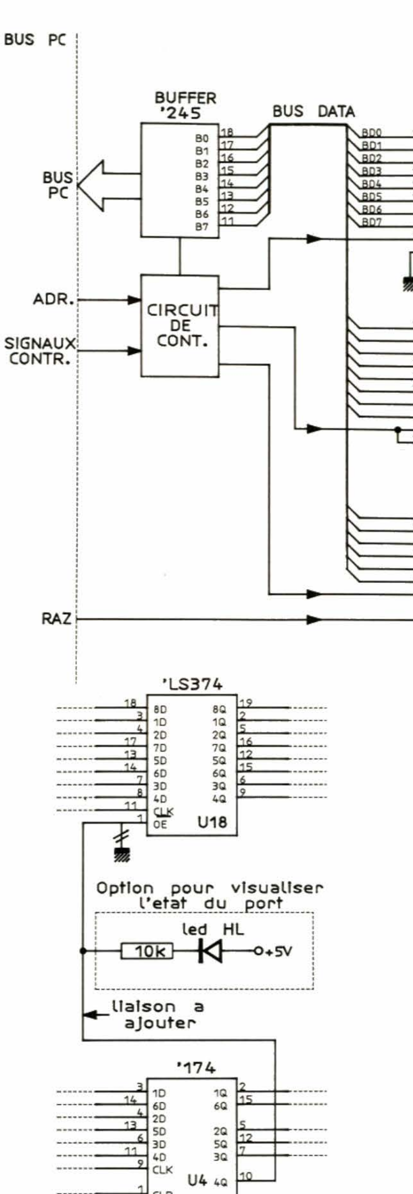


Figure 1

Figure 2

du port.
Il faut utiliser une LED haute luminosité, elle pourra être installée sur la face arrière. Elle sera allumée lorsque le port est configuré normalement, c'est-à-dire en sortie (après un RESET, à la mise sous tension, pendant une impression...).

Le programme en BASIC

Ce petit programme essaie de configurer les interfaces imprimante en entrée.
Comme il existe deux espaces d'adressage, deux essais sont effectués.
Il commence par initialiser les adresses correspondant au premier port, puis effectue une tentative de détection, il doit relire les données mises sur le port DATA de l'interface, si c'est le cas, l'interface est présente, dans le cas contraire on considère que l'interface n'existe pas à cette adresse et le logiciel va faire une tentative sur le second espace d'adressage possible.
Si l'interface a été trouvée, nous positionnons successivement à 1 les trois derniers bits du registre de contrôle ; pour chaque essai on teste si le port est configuré en lecture seule (désactivation du 74LS374). Si un des cas est positif, le message indiquant le bit à utiliser du registre de contrôle est envoyé.
Quel que soit le résultat, le port est repositionné en sortie en mettant les trois derniers bits à 0.

Le protocole

A priori, il n'en existe aucun de normalisé, à chacun son protocole.
Si l'on veut utiliser au mieux cette interface, il est souhaitable de respecter quelques règles.
Il faut se réserver une ligne pour indiquer au périphérique la configuration active. A la mise sous tension, et en mode normal, cette ligne devra indiquer une configuration sortante afin de ne pas avoir de problème avec le périphérique qui peut lui aussi ÉMETTRE des données.
Une gestion sous interruption est plus souple qu'une gestion par échantillonnage (polling), il faut prévoir ce type de fonctionnement.
La mise à 1 du bit 4 du registre de contrôle autorise la gestion sous interruption, dans ces conditions un front descendant sur ACK génère une interruption.
Il faut aussi modifier le vecteur, écrire le programme etc.
Cette seconde règle doit être prévue, même si, pour des applications simples, l'utilisation du polling est plus simple à mettre en œuvre.

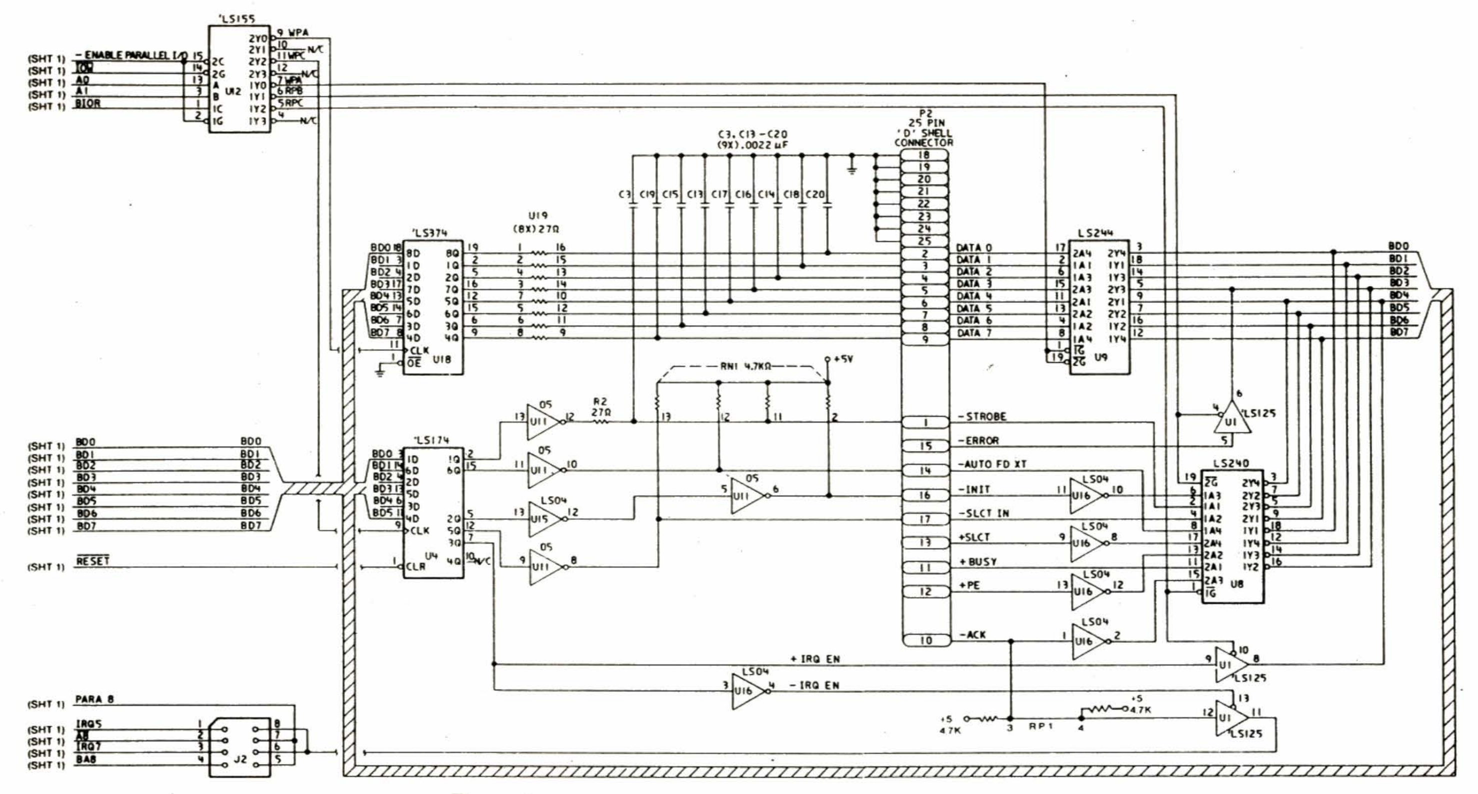


Figure 3

Enfin, il faut qu'à la fin du traitement, le programme reconfigure le port dans le mode imprimante, en cas de sortie brutale du programme, la LED éteinte indiquera que le port n'est pas configuré pour l'imprimante.
Si l'on ne souhaite pas déconnecter son imprimante, la solution la plus souple est certainement celle qui consiste à avoir deux cartes imprimantes installées, l'une pour l'imprimante, l'autre pour les périphériques bidirectionnels qui utiliseront cette interface.

Mise en garde

Il existe des PC qui ne sont pas de vrais compatibles (clone) PC à 100 %, la sortie imprimante n'utilise pas le principe évoqué dans cet article, l'auteur ne peut rien proposer pour ce type de machine, une réalisation matérielle sera sans doute nécessaire.

En conclusion

Cette interface bidirectionnelle, de coût ridicule, va permettre de simplifier les interfaces et le logiciel de gestion des montages "bizarres". La vitesse d'échantillonnage sur cette entrée ne sera limitée que par la vitesse du PC et le langage utilisé...

X. Fenard



```
10 PRINT "1:Recherche, 2:Recherche puis Lecture en boucle"
20 INPUT V
25 VI=0
30 GOSUB 100
35 IF V=1 THEN END
40 IF VI=1 GOTO 60
45 IF VI=2 GOTO 80
50 END
60 GOSUB 1010
61 GOTO 90
80 GOSUB 1110
90 PRINT INP(PRND%)
95 GOTO 90
100 GOSUB 1000:REM adresse display
105 L%= (INP(CTRL%) AND &H1F)
106 OUT CTRL%,L%
110 GOSUB 2000
120 GOSUB 2055:REM compte rendu
130 IF N= 0 GOTO 200
140 GOSUB 3000
150 IF N=0 THEN VI=1 :REM VI=1 detectee
200 GOSUB 1100:REM adresse PRINTER
205 L%= (INP(CTRL%) AND &H1F)
206 OUT CTRL%,L%
210 GOSUB 2000
220 GOSUB 2055:REM compte rendu
230 IF N=0 GOTO 250
240 GOSUB 3000
250 IF N=0 THEN VI=2 :REM VI=2 detectee
250 RETURN
999 END
1000 REM Adresse pour le port Display
1001 PRINT "Recherche sur le port DISPLAY en 3BC"
1009 REM en lecture/ecriture
1010 PRND% = &H3BC: CTRL% = &H3BE
1019 REM en lecture seulement
1020 STATI = &H3BD
1030 RETURN
1100 REM Adresse pour le port printer
1101 PRINT "Recherche sur le port PRINTER en 378"
1109 REM en lecture/ecriture
1110 PRND% = &H378: CTRL% = &H37A
1119 REM en lecture seulement
1120 STATI = &H379
1130 RETURN
2000 REM Detection de CARTE
2010 OUT PRND%,&H55
2020 IF (&H55 <>INP(PRND%)) GOTO 2050
2030 OUT PRND%,&HAA
2040 IF (&HAA <>INP(PRND%)) GOTO 2050
2045 N=1:RETURN
2050 N=0:RETURN
2055 IF N=0 goto 2070
2060 PRINT "Interface imprimante detectee":RETURN
2070 PRINT "Interface imprimante pas detectee a cette adresse":RETURN
3000 PRINT " RECHERCHE d'une configuration entree "
3005 L%= ( (INP(CTRL%) AND &H1F) +32)
3010 K=5 : OUT CTRL%,L%
3020 GOSUB 2000
3030 IF N=0 GOTO 3090
3035 L%= ( (INP(CTRL%) AND &H1F) +64)
3040 K=6 : OUT CTRL%,L%
3050 GOSUB 2000
3060 IF N=0 GOTO 3090
3065 L%= ( (INP(CTRL%) AND &H1F) +128)
3070 K=7 : OUT CTRL%,L%
3080 GOSUB 2000
3085 IF N=0 GOTO 3090
3087 PRINT "PAS de configuration entree possible"
3088 L%= (INP(CTRL%) AND &H1F)
3089 OUT CTRL%,L% : RETURN
3090 PRINT "Le bit ";K;" du controle mis a 1 configure le port en entre"
3092 IF V=2 THEN RETURN
3095 L%= (INP(CTRL%) AND &H1F): OUT CTRL%,L% :RETURN
```

Le programme en BASIC.

PEDALE INTERRUPTEUR "TREADLITE"



Pédale interrupteur à double détente (Mi-course et fin de course). (2 micro-switches 7A/250V). Avec patin anti-dérapant. Matériel professionnel pour usage intensif. Idéale pour télécommande de perceuse, etc...

La pédale .... 113.3831 **100,00 F**

FILTRE SECTEUR 10 A

Matériel professionnel. Entrée sur embase CEE. Sorties sur cosses FAST-ON.

Le filtre... 113.3830 **110,00 F**

VENTILATEUR PROFESSIONNEL PAPST



220 V. Dim. 80 x 80 x 38 mm. Parfaitement silencieux. (24 dBA). Sans comparaison avec les ventilateurs standard.

Le ventilateur ... 113.3813 ~~250,00 F~~ **140,00 F**

RELAIS STATIQUE 10A/240 V



Tension de commande : 3,8 à 28 V DC. Commutation au zéro de tension. Matériel professionnel. Sorties sur fast-on.

Le relais statique .... 113.3785 **100,00 F**

MOTEUR PAS A PAS BIPOLAIRE



De puissance. 200 pas/tour. 1 A / phase - 4 fils. Fourni avec fiche technique détaillée.

Le moteur .... 113.4302 **190,00 F**

TOURNEVIS DE PRECISION



Set de 6 tournevis pour l'électronique. 4 à lame + 2 cruciformes. Embout au molybdène. Manche ergonomique avec bout rotatif. Fourni avec support de rangement.

Le set de 6 tournevis .... 113.3784 **66,00 F**

CARTES ALIMENTATION EN KIT

Qualité professionnelle. Tensions de sorties redressées, filtrées, régulées. Sorties flottantes. Voyants LED de contrôle. E/S sur borniers à vis. Kits fournis complets avec c. imp. Dim. : 115 x 95 x 40 mm

Alim. ± 12 V / 0,6 A ou 24 V / 0,6 A	Le kit	113.8742	**155,00 F**
Alim. ± 5 V / 1,1 A ou 10 V / 1,1 A	Le kit	113.3711	**175,00 F**
Alim. 5 V + 12 V / 0,6 A	Le kit	113.8743	**155,00 F**
Alim. 5 V + 8 V / 1,1 A	Le kit	113.3708	**175,00 F**

3616 SELECTRO

VOILA LE CODE D'APPEL DU SERVEUR MINITEL SELECTRONIC que vous pouvez consulter à partir du 4 juin 1992 !

Il comprend :
- Un service d'assistance et de renseignements techniques
- Un forum **BUS-I²C** et **COMM'net**
- Un service des dernières nouveautés et promotions
- Un service de petites annonces classées. Etc...



C.I.F. et SELECTRONIC SE SONT UNIS POUR RESOUDRE VOTRE PROBLEME DE REALISATION DE CIRCUITS IMPRIMES...

Et vous proposent de faire l'acquisition de votre "unité de fabrication" de circuits à des conditions particulièrement avantageuses !

OFFRE N°1



Vous commandez : 1 MACHINE A INSOLER MI-1016 **2200,00 F**
1 MACHINE A GRAVER BB-4 **1495,00 F**
TOTAL TTC **3695,00 F**

NOUS VOUS OFFRONS :
- 1 jerrycan 5 l de perchlo suractivé
- 2 sachets de détachant pour perchlo
- 1 sachet de 10 gants de protection
- 1 bac AR-23
- 6 plaques EPOXY 1 face 200x300 présensibilisé
- 10 sachets de révélateur positif
- 1 flacon 1/2 litre étain chimique
- 1 stylo DALO

(Ensemble d'une valeur de 691,70 F TTC)

C.I.F CIRCUIT IMPRIMÉ FRANÇAIS

LE TOUT OFFRE N°1.................. 113.3750 **3695,00 F**

Forfait PORT (Transporteur) et EMBALLAGE en sus 150,00 F

OFFRE N°2



Vous commandez : 1 MACHINE A INSOLER EN KIT BC-6 **1068,00 F**
1 MACHINE A GRAVER BB-2 **1300,00 F**
TOTAL TTC **2368,00 F**

NOUS VOUS OFFRONS :
- 3 sachets de perchlo en poudre
- 2 sachets de détachant pour perchlo
- 1 sachet de 10 gants de protection
- 6 plaques EPOXY 1 face 150x200 présensibilisé
- 3 plaques EPOXY 1 face 100x150 présensibilisé
- 10 sachets de révélateur positif
- 1 stylo CIF
- 1 bac AR-23

(Ensemble d'une valeur de 430,00 F TTC)

C.I.F CIRCUIT IMPRIMÉ FRANÇAIS

LE TOUT OFFRE N°2.................. 113.3640 **2368,00 F**

Forfait PORT (Transporteur) et EMBALLAGE en sus 150,00 F

Nous avons la solution !

TOUJOURS DES OPPORTUNITES ET PROMOTIONS CHEZ SELECTRONIC !
Envoi de notre lettre d'informations sur simple demande.

Selectronic
la passion de l'électronique !

CONDITIONS GENERALES DE VENTE : *Voir nos publicités annexes.*

VENTE PAR CORRESPONDANCE BP 513 - 59022 LILLE CEDEX

TEL : 20 52 98 52 - FAX : 20 52 12 04

RP

ISI, logiciel de calculs techniques

Conçu et distribué par IB TECHNIC (voir annonceurs), ISI est un logiciel pour PC, regroupant diverses tables et calculs. Son faible coût (289 F) nous a donné envie d'en savoir plus à son sujet, et il s'est avéré que c'était un outil bien fait : plaisant et utile, et qu'il méritait sa place sur tout disque dur. A noter qu'il ne s'adresse pas uniquement aux électroniciens : les mécaniciens y trouveront aussi largement leur compte.

Les possibilités, fort nombreuses, peuvent se répartir en 5 groupes :
1 - Les modules de conversion.
2 - Les tables.
3 - Un simulateur logique.
4 - Les calculs électroniques.
5 - La géométrie et la mécanique.
Certains de ces groupes sont actifs (calculs), d'autres servent de mémento pour des données qu'il est toujours utile d'avoir à portée de main, telles les tables trigonométriques, les codes ASCII, les densités des matériaux, etc...

La convivialité du logiciel est telle que dans la majorité des cas, un coup d'œil sur le manuel n'apportera que les finesses spécifiques à chaque groupe. Seul le simulateur logique n'est pas "évident" : c'est d'ailleurs écrit d'emblée dans le mode d'emploi.
Il méritera un apprentissage minimum - ce que nous n'avons pas fait, n'ayant pas de problème particulier à résoudre -, et puis il y avait tellement d'autres calculs passionnants comme la surface d'une calotte sphérique, le poids d'un tore en cuivre, etc... que nous avons préféré "jouer" avec ISI.
L'usage des menus déroulants n'est pas systématique, et dans certains groupes il est possible d'accéder directement à une donnée à modifier, pour obtenir immédiatement le ou les résultats sur un seul et même écran.
Voyons donc d'un peu plus près chaque module.

Modules de conversion

Trois sous-groupes sont proposés : conversions de TAILLE, de BASE ou de TEMPERATURE.
Pour la TAILLE, on obtient instantanément les équivalences en millimètres, centimètres, pouces et pieds, d'une donnée entrée en quelque unité que ce soit.
En conversion de BASE, le principe est le même pour BINAIRE, DECIMAL, OCTAL et HEXADECIMAL. Ainsi, si vous tapez 179 en décimal, vous obtenez 10110011 en binaire, 263 en octal et B3 en hexa. Ce qui est très intéressant par rapport à une calculette, c'est que données et résultats sont tous présents à l'écran. Si on entre une donnée en binaire et que l'on demande la conversion hexa sur une calculette, seul le résultat est donné. Avec ISI, tous les éléments restent visibles, et c'est très utile : une erreur de saisie est tellement vite faite !
La conversion de température assure la traduction CELSIUS, FAHRENHEIT.

Tables

Trois sous-groupes sont encore proposés : table des caractères ASCII, des densités et trigonométrique.
Pour les caractères ASCII, on peut taper le code désiré, ou encore se promener avec les flèches "haut/bas" dans la liste globale (c'est aussi rapide).
A chaque ligne, on obtient :
– le caractère ASCII sous sa forme graphique.
– Son équivalence en décimal.
– En hexa.
– En binaire.
– En BCD.
La table des densités offre une liste de matières avec leur densité. Il est possible de modifier certaines données et de créer de

nouveaux produits. Cette table servira par ailleurs aux calculs de surfaces, hé oui ! (voir plus loin), et de volumes.
Enfin la table trigonométrique permet la visualisation :
- des données en degrés décimaux ou hexadécimaux,
- des sinus ou cosinus,
- des radians ou des facteurs de pi,
- des angles complémentaires,
- des angles supplémentaires,
- des tangentes et cotangentes,
- des sinus ou cosinus.

Simulateur logique

Nous vous l'avons dit (et le manuel le précise), ce n'est pas le module le plus évident. Il faudra lire attentivement la notice et se confronter de préférence à un problème réel dont la solution sera présumée connue.
Il travaille sur 8 octets pour lesquels on peut opter pour les fonctions AND, OR, XOR, NAND, NOR, XNOR et bien entendu pour des valeurs de 00 à FF. Le but est de vérifier le fonctionnement d'un montage, ou de retrouver - à partir d'un résultat espéré - la ou les configurations correctes.
Les lycées et collèges trouveront sans aucun doute un intérêt à ce module pour son aspect éducatif.

Calculs électroniques

Ils ne sont pas très nombreux, ni très passionnants. IB TECHNIC nous pardonnera cette critique, mais si on pense disposer d'une "calculatrice spécialisée électronique" on risque de se plaindre ! Les possibilités sont limitées à des calculs R, C, RC primaires, sur la série E12 exclusivement, et ne tiennent pas compte de la tolérance des composants. Les calculs d'alimentations sont aussi très simplifiés.
Mais il semblerait que nous ne soyons en présence que d'une première mouture, et si certaines options sont très complètes et puissantes, d'autres balbutient encore. Il eût peut-être été plus judicieux d'attendre que ces dernières soient un peu plus matures, mais c'est ainsi, et sûrement provisoire !

Calculs de surfaces et volumes

Contrairement aux deux précédents modules, ce dernier nous a passionné car excessivement puissant.
Pour les surfaces, il ne propose pas moins de 30 formes différentes, avec une option "épaisseur" (!) liée au module densité (patience...)
Cinq résultats sont donnés : surface réelle en mm², corrigée en mm², cm², dm² et m².
Mais une surface dotée d'une épaisseur serait plutôt un volume ! C'est vrai, mais IB TECHNIC a choisi de faire la différence entre le volume de matière effective d'un objet (auquel on peut appliquer une densité), et le volume d'une enveloppe.
Il serait plus facile de prendre un exemple : soit une balle de ping-pong. Le volume apparent (externe) de la sphère et le volume de matière utile (la "peau") - auquel on peut affecter une densité connue (celle de la matière : celluloïd ?) - sont deux résultats à différencier.
Vu simplement de l'extérieur, la sphère est mesurable en volume, et surface de peau : volume de la sphère, surface de la dite sphère. Mais la balle est (celluloidement... parlant) une masse mesurable. Son poids est alors lié à la densité de la matière et au volume réel de l'objet, soit surface-épaisseur.
Donc si vous voulez connaître immédiatement le poids d'une coupelle d'acier de rayon X, de base Y et d'épaisseur Z, c'est au calcul des surfaces qu'il faudra faire appel.
Le calcul des volumes propose 24 choix dont le tonneau, et le "quadrangulaire à deux bases" dit "tas de sable".
Pour le volume et le poids d'un anneau sphérique creux, il faudra faire une soustraction : anneau sphérique externe - anneau sphérique interne.
Si il fallait absolument regretter quelque chose, ce serait par exemple que la formule de calcul soit inaccessible, mais il y a des ouvrages pour cela.

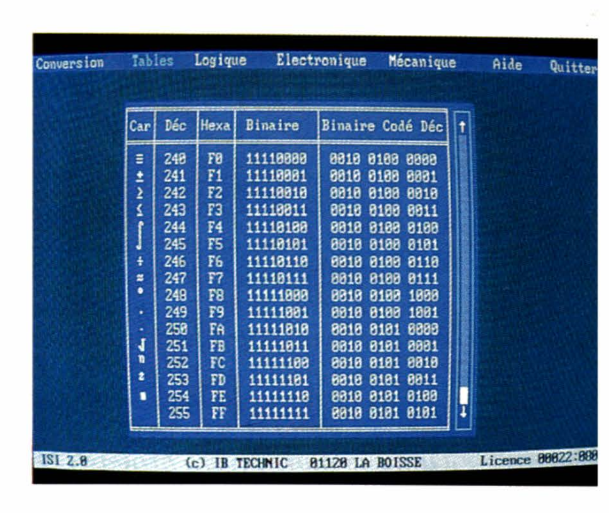
CONCLUSION

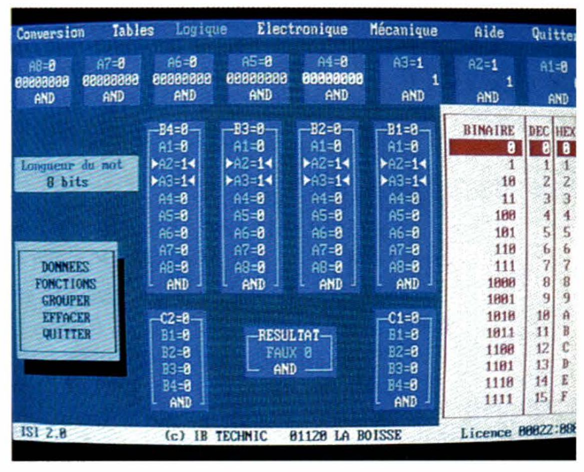
ISI est un excellent logiciel à un prix très sympa, convivial et simple à utiliser, même si parfois on en voudrait encore (toujours !) plus.
Parmi les autres produits IB TECHNIC, nous vous présenterons prochainement LORIMEN : logiciel de gestion de nomenclatures.
A suivre donc...

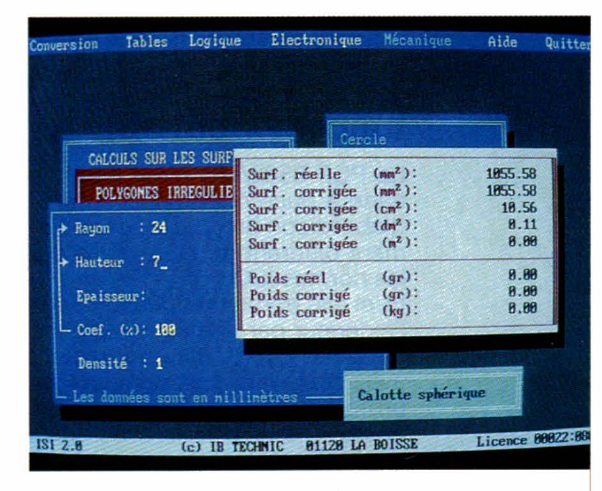
Jean ALARY



IB Technic
BP 602
119, RN, La Boisse
01126 Montluel Cedex
Tél. : 78.06.44.90

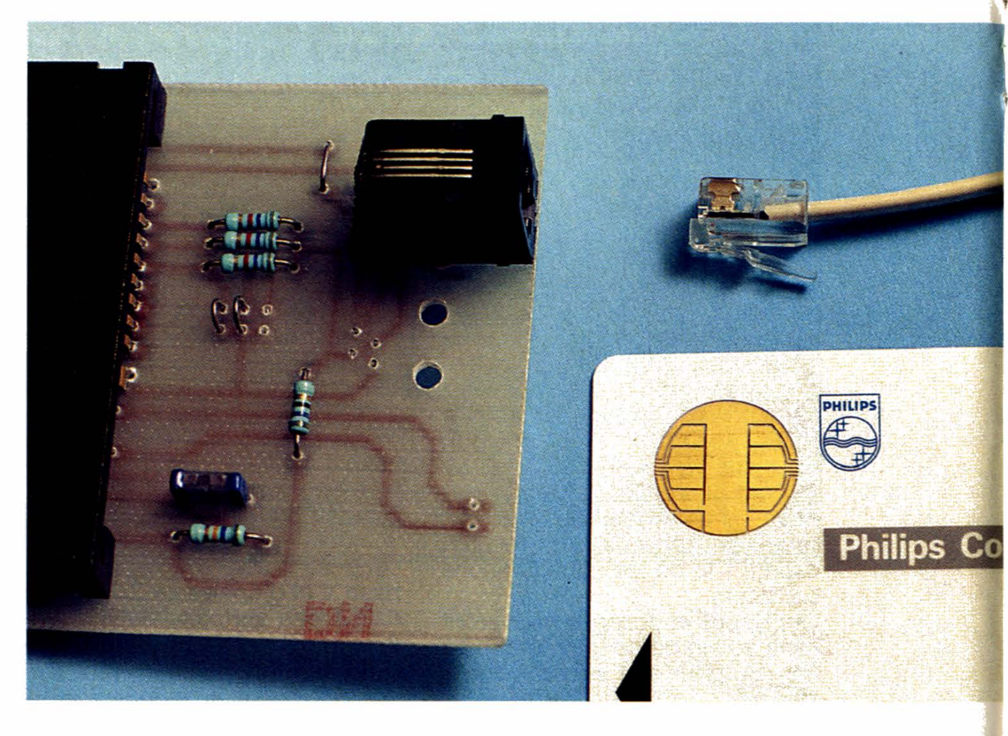






Un lecteur-programmateur de carte à puce I2C

Parmi la gamme en pleine extension des circuits intégrés compatibles avec le bus I2C, il existe toute une variété de mémoires, à commencer par les RAM et les EEPROM. Nos lecteurs connaissent bien l'EEPROM PCF 8582, déjà utilisée dans divers projets I2C décrits dans cette revue. Cette mémoire non volatile et réinscriptible de 256 octets (soit 2 kbits) répond à bien des critères exigés pour les applications "carte à mémoire". Il n'en fallait pas plus pour que PHILIPS développe une version encartée de ce composant, la PCF 8582/MC. Dès lors, il suffit d'un montage très simple et d'un peu de logiciel pour mettre en œuvre une application "carte à puce" sur n'importe quel système I2C, domotique ou autre : dossier portable, contrôle d'accès, etc.



UNE CARTE À PUCE I2C

La PCF 8582/MC n'est pas une carte à puce comme les autres : d'abord son brochage est totalement différent de celui défini par les normes ISO !
Et si elle communique bien avec l'extérieur par une ligne série (on ne peut guère faire autrement avec seulement huit contacts), le protocole de cette liaison n'a pas grand chose à voir avec celui applicable aux cartes courantes.
Bref, la carte est accessible à travers le protocole I2C, point final.
Bien entendu, cela suppose des lecteurs et programmateurs spécifiques, mais présentant l'avantage d'être extrêmement simples dès lors qu'ils sont reliés à un "maître" I2C (microcontrôleur ou ordinateur adapté en conséquence).
Prenons le cas d'une installation domotique existante, architecturée autour d'un bus I2C : il suffira de raccorder le lecteur de cartes par quatre fils et d'écrire un peu de logiciel pour mettre en place une sécurisation par carte à puce des fonctions "sensibles".
Mais avec une capacité de 256 octets d'EEPROM, la carte PCF 8582/MC peut faire beaucoup mieux que de servir de simple clef électronique : on peut en faire un véritable "dossier portable" réinscriptible, servant par exemple à transporter des résultats de mesure ou au contraire des paramètres de réglage entre différents appareils.
Moyennant l'installation d'équipements adéquats à l'accueil et sur les stands, cette carte ferait un excellent "badge" pour les visiteurs de salons professionnels : 256 caractères suffisent amplement pour enregistrer les coordonnées complètes du porteur, plus un certain nombre de renseignements nécessaires aux organisateurs.
Ce n'est finalement guère qu'une question d'imagination...
La **figure 1** reproduit le schéma synoptique général de toutes les mémoires de la famille PCF 8582, applicable notamment à notre carte à puce.
On notera bien qu'il s'agit d'une "carte à mémoire simple", dépourvue de tout dispositif de sécurité et pouvant donc être librement lue, copiée, ou repro-

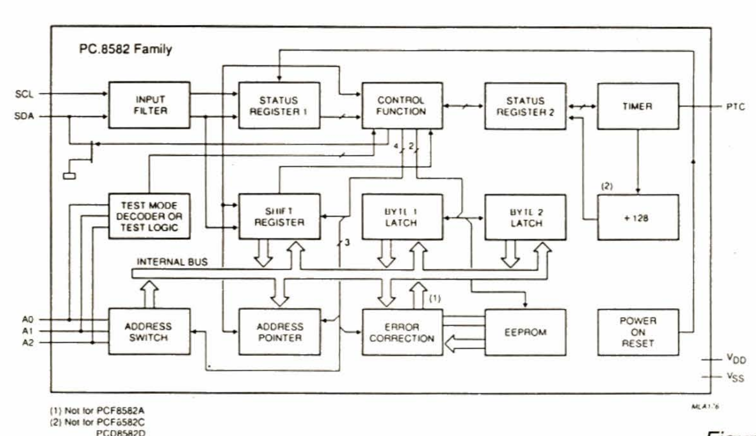


Figure 1

grammée par tout détenteur d'un lecteur-programmateur compatible. Pas question donc de s'en servir pour des applications critiques sur le plan de la sécurité, ou pour de la monétique !

Le brochage représenté à la **figure 2** est celui de la version présentée en boîtier DIL à 8 broches : on verra qu'il est exactement inverse de celui de la carte, ce qui est logique si on réfléchit à la position occupée par la puce dans le micromodule de la carte.

RÉALISATION DU LECTEUR-PROGRAMMATEUR

En principe, le rôle du lecteur-programmateur devrait se limiter à de la connectique reliant les contacts de la carte aux quatre fils du bus I2C.

Dans la pratique, il faut ajouter quelques composants pour faire fonctionner l'horloge interne de la mémoire et pour lui attribuer une adresse I2C complète.

Bien évidemment, la carte est alimentée par le bus lui-même (+ 5 V et masse), ce qui ne pose aucun problème du fait de sa très faible consommation.

Comme la plupart des mémoires I2C, la PCF 8582 (MC ou pas) s'est vue affecter l'adresse de base 1010 (ou A en hexadécimal).

Le dernier bit de l'adresse complète servant à spécifier si on effectue une lecture (1) ou une écriture (0), trois bits restent disponibles pour donner à la carte une adresse complète choisie parmi huit possibles, déduction faite bien sûr de celles attribuées aux éventuelles autres mémoires du système.

Le tableau de la **figure 3** récapitule ces possibilités, sous la double forme des niveaux logiques appliqués aux broches A_0, A_1, A_2 de la mémoire et d'adresses hexadécimales compatibles avec la syntaxe du BASIC du COMMnet, microcontrôleur particulièrement bien adapté à la gestion de ce montage.

BASE : 1010 (Ah)			Adresse i2c	
A_2	A_1	A_0	0A0h	0A1h
0	0	1	0A2h	0A3h
0	1	0	0A4h	0A5h
0	1	1	0A6h	0A7h
1	0	0	0A8h	0A9h
1	0	1	0AAh	0ABh
1	1	0	0ACH	0ADh
1	1	1	0AEh	0AFh

Figure 3

On se souviendra alors que, dans le COMMnet, se trouve une mémoire à horloge temps réel dont l'adresse est 0A0h ou 0A1h : il convient évidemment d'éviter cette combinaison au niveau du lecteur de cartes !

Pour notre part, nous avons choisi l'adresse 0A8h ou 0A9h (A0 et A1 à la masse, A2 au niveau haut).

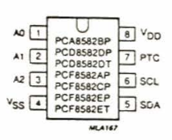
On aboutit ainsi au schéma de la **figure 4** :

Trois résistances de tirage rappellent au niveau haut les trois entrées d'adresse, ce qui fait qu'il suffit de relier à la masse celles devant recevoir un niveau bas : cela peut se faire par des fils soudés, des points de soudure, ou des cavaliers mobiles sur deux rangées de picots (barrettes sécables).

Un réseau RC (56 kΩ et 2,2 nF) sert à fixer à 21 ms la durée du cycle de programmation, paramètre important pour une EEPROM.

La résistance de 56 Ω, pour sa part, sert à limiter le courant d'alimentation si, lors de l'insertion de la carte, un court-circuit devait se produire entre la masse et le VDD. Avec le brochage utilisé, cela ne risque guère de se produire, mais sait-on jamais ce qui pourra être introduit tôt ou tard dans la fente du lecteur ?

Le circuit imprimé de la **figure 5** a été dessiné conformément au brochage désormais normalisé des connecteurs de carte à puce, qui correspond à celui du dernier modèle d'ITT-CANNON disponible au détail chez SELECTRONIC.



SYMBOL	PIN	DESCRIPTION
A0	1	address input
A1	2	address input
A2	3	address input
V_{SS}	4	ground
SDA	5	serial data
SCL	6	serial clock
PTC	7	can be connected to V_{DD} or left open-circuit
V_{DD}	8	positive supply voltage

Figure 2

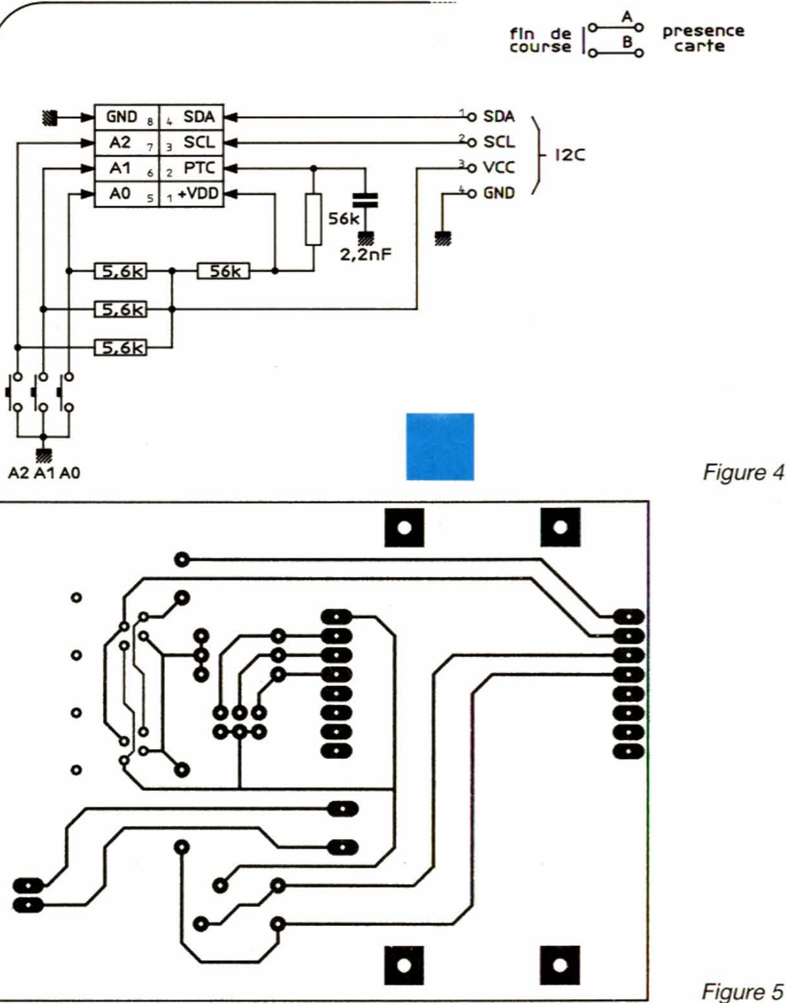


Figure 4

Figure 5



Le raccordement au bus I2C est prévu par un jack modulaire à 4 positions et 4 contacts, monté en parallèle avec un second permettant le raccordement simultané d'autres accessoires I2C.
Cette disposition a été décidée arbitrairement, en l'absence de toute norme en la matière. Elle présente l'avantage de permettre l'exécution des interconnexions à l'aide de câbles mâle-mâle non croisés, faciles à réaliser en n'importe quelle longueur par sertissage de deux fiches aux extrémités d'un câble spécial.
Précisons d'ailleurs qu'il existe des embases murales pour ces fiches, et que les prix de tous ces connecteurs sont fort minimes : ils équipent en effet à peu près tous les téléphones américains en lieu et place de nos volumineux "conjoncteurs", au moins huit fois plus chers...
Mais bien entendu, tout autre mode de raccordement est possible : il suffit de souder quatre fils dans les pastilles voulues !
L'implantation de la **figure 6** comporte un strap fixe qu'il conviendra de ne pas oublier, et deux autres servant à fixer l'adresse I2C du lecteur conformément aux conventions de la **figure 3**.
Deux pastilles sont aussi prévues pour accéder au contact de détection de carte, nécessaire ou non selon les applications.

EXPLOITATION LOGICIELLE

Le programme à écrire pour gérer ce montage dépend évidemment de l'application à mettre sur pied, et du "maître" I2C utilisé. Il pourra s'agir, par exemple, d'un logiciel en assembleur pour un monochip I2C.
Pour notre part, nous avons expérimenté notre lecteur-programmateur sur le COMMnet de SOCAVEL, distribué par SELECTRONIC.

Ce choix présente l'avantage de permettre une programmation en BASIC, et donc à la portée de tous nos lecteurs.
Assez proche du BASIC INTEL utilisé sur le 8052 AH BASIC, le langage du COMMnet possède des instructions supplémentaires, et notamment celles permettant de dialoguer de façon entièrement "transparente" avec le bus I2C.
Le programme de la **figure 7**, d'une simplicité biblique, sert à lire entièrement la carte dont les 256 octets s'affichent en décimal sur l'écran du terminal relié au COMMnet (MINITEL ou PC, par exemple) sous la forme que montre la **figure 8**.
La première instruction du programme (ligne 10) sert à mettre à zéro le compteur d'adresses de

```
READY
]list
10       IIC 0A8H,0!
12       DIM B(16)
15       POP Z
20       FOR L=1 TO 16
25       IIC 0A9H,16
26       POP Z
30       FOR C=1 TO 16
40       POP A
50      B(C)=A
60       NEXT C
70       FOR C=1 TO 16
80       PRINT B(17-C),
90       NEXT C
100      PRINT
110      NEXT L

READY
]
```

Figure 7

```
80  65  84  82  73  67  75  170  85  85  255  85  85  85  85  85
170 170 170 170 170 170 170 170 85 85 85 85 85 85 85 85
170 170 170 170 170 170 170 170 85 85 85 85 85 85 85 85
170 170 170 170 170 170 170 170 85 85 85 85 85 85 85 85
170 170 170 170 170 170 170 170 85 85 85 85 85 85 85 85
170 170 170 170 170 170 170 170 85 85 85 85 85 85 85 85
170 170 170 170 170 170 170 170 85 85 85 85 85 85 85 85
170 170 170 170 170 170 170 170 85 85 85 85 85 85 85 85
170 170 170 170 170 170 170 170 85 85 85 85 85 85 85 85
170 170 170 170 170 170 170 170 85 85 85 85 85 85 85 85
170 170 170 170 170 170 170 170 85 85 85 85 85 85 85 85
170 170 170 170 170 170 170 170 85 85 85 85 85 85 85 85
170 170 170 170 170 170 170 170 85 85 85 85 85 85 85 85
170 170 170 170 170 170 170 170 85 85 85 85 85 85 85 85
170 170 170 170 170 170 170 170 85 85 85 85 85 85 85 85
170 170 170 170 170 170 170 170 85 85 85 85 85 85 0 85

READY
]
```

Figure 8

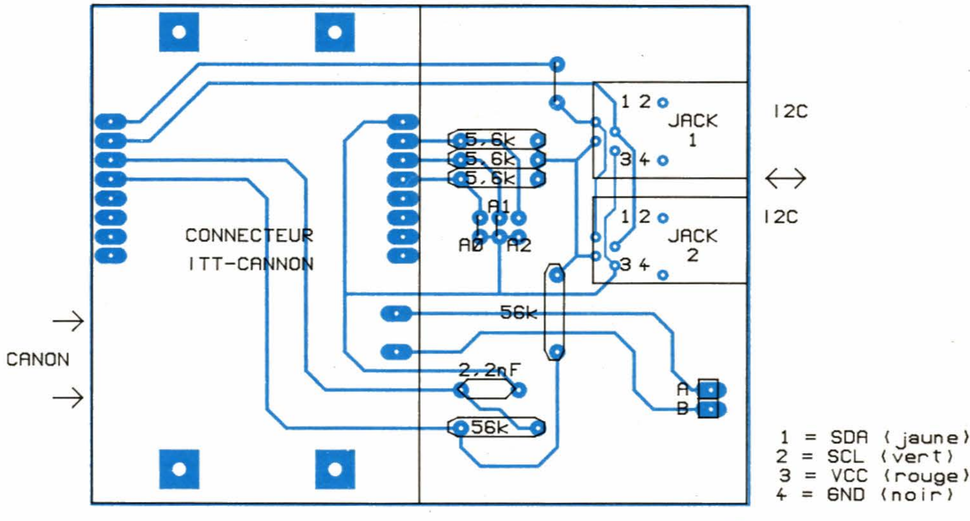


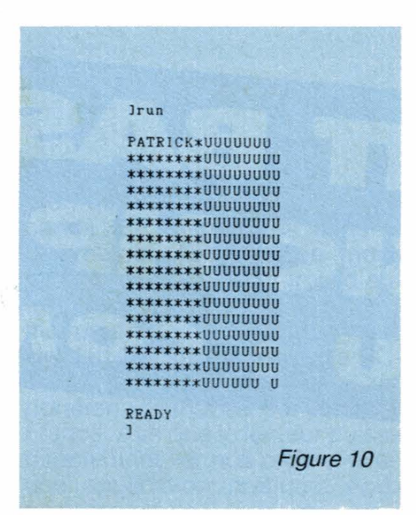
Figure 6

la carte, qui s'incrémentera automatiquement par la suite, au fil des opérations de lecture (ou d'écriture s'il y en avait).
En principe, la carte se remet à zéro de façon autonome lors de sa mise sous tension, mais deux précautions valent mieux qu'une.
Une variante de cette instruction (avec un argument différent de 0), permettrait d'accéder directement à n'importe quelle adresse de la carte.
On notera que les valeurs lues par l'instruction IIC 0A9H,16 transitent par la "pile", et se trouvent dépilées dans l'ordre inverse de celui de leur lecture. Le bon ordre est rétabli en passant par un tableau intermédiaire en mémoire, DIM (16).
Après chaque opération de lecture ou d'écriture, le premier octet dépilé (Z) est le code d'erreur rendant compte de la réussite ou de l'échec de la transaction.
Nous ne le traitons pas dans ces petits programmes d'illustration, mais dans un véritable logiciel applicatif, il faudrait le faire.
Ainsi pourraient être détectées des situations anormales comme des tentatives de dialogue sans carte dans le lecteur, ou en présence d'une carte non-I2C (télécarte par exemple).
Le manuel du COMMnet contient toutes les informations nécessaires à l'interprétation très fine de ces codes d'erreur.
Le programme de la **figure 9**,

```
READY
]list
10        IIC OA8H,O!
12        DIM B(16)
15        POP Z
20        FOR L=1 TO 16
25        IIC OA9H,16
26        POP Z
30        FOR C=1 TO 16
40        POP A
50       B(C)=A
60        NEXT C
70        FOR C=1 TO 16
80        PRINT CHR(B(17-C)),
90        NEXT C
100       PRINT
110       NEXT L

READY
]
```

Figure 9



```
]run

PATRICK*UUUUUUU
********UUUUUUUU
********UUUUUUUU
********UUUUUUUU
********UUUUUUUU
********UUUUUUUU
********UUUUUUUU
********UUUUUUUU
********UUUUUUUU
********UUUUUUUU
********UUUUUUUU
********UUUUUUUU
********UUUUUUUU
********UUUUUUUU
********UUUUUUUU
********UUUUUU U

READY
]
```

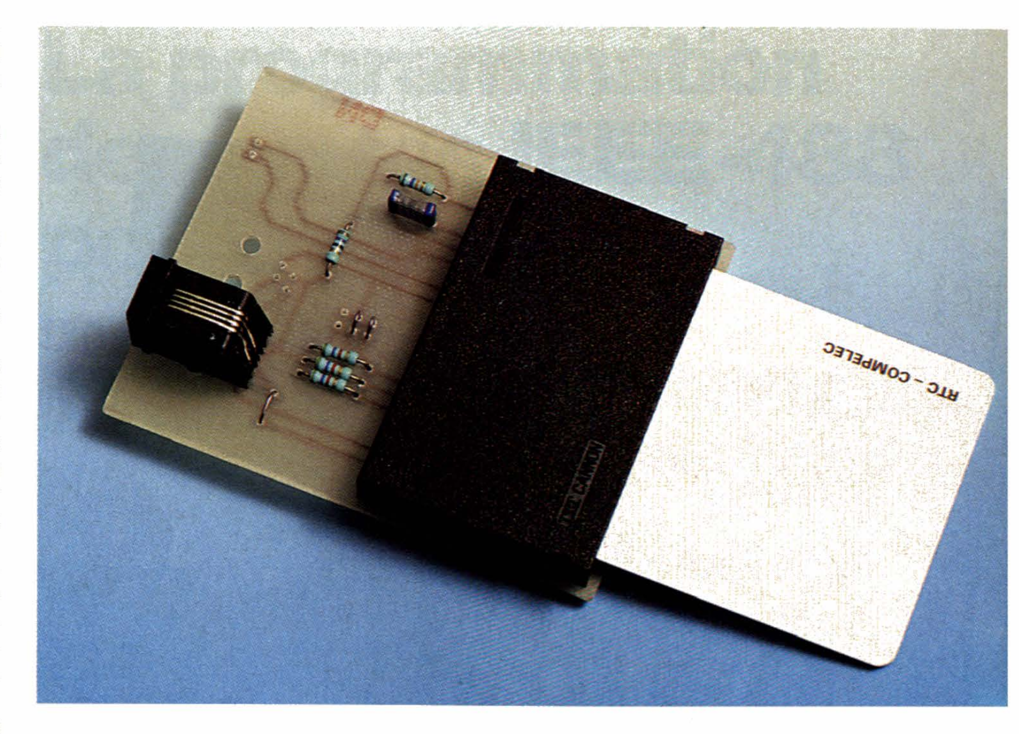
Figure 10

dérivé du précédent, affiche le contenu de la carte sous la forme de 256 caractères ASCII, selon la présentation de la **figure 10** : cela peut être utile lorsque la carte sert à conserver des données alphanumériques.
Avec le programme de la **figure 11**, nous abordons l'écriture dans la carte, écriture qui n'a rien de définitif puisque nous sommes en présence d'une

```
READY
]list
10        INPUT "ADRESSE ?",A
20        IF A>255 THEN  END
30        IIC OA8H,A!
40        POP Z
50        IIC OA9H,1
60        POP Z
70        POP D
80        PRINT A,":",D,CHR(D)
90        INPUT "NOUVELLE DONNEE ?",C
100       IF C>255 THEN  END
110       IF (C=D) THEN 10
120       IIC OA8H,A,C!
125       POP Z
130       GOTO 10

READY
]
```

Figure 11



EEPROM à effacement automatique avant écriture : tout octet existant peut être remplacé par n'importe quel autre, jusqu'à concurrence de 10 000 fois environ.
L'utilisateur doit spécifier une adresse, qui est d'abord lue. Le contenu affiché peut alors être modifié en tapant simplement la nouvelle valeur (en décimal).
A tout moment il est possible de sortir du programme en entrant une donnée supérieure à 255 qui, évidemment, ne sera pas écrite en mémoire.

Patrick GUEULLE



«Si les autres y étaient parvenus, lui aussi y arriverait. Il saisit la disquette, l'inséra délicatement dans le lecteur de son PC. Des gouttes de transpiration coulaient le long de son dos. Il était si proche du but. Soudain une vague d'allégresse le submergea. Le programme DEMO défilait sous ses yeux. Il avait réussi. Son PC était devenu aussi puissant qu'un HP 9000. Il parlait le même langage. Il empoigna son manuel HT-BASIC et le serra contre sa poitrine. Lui aussi disposait maintenant d'un calculateur technique ...»

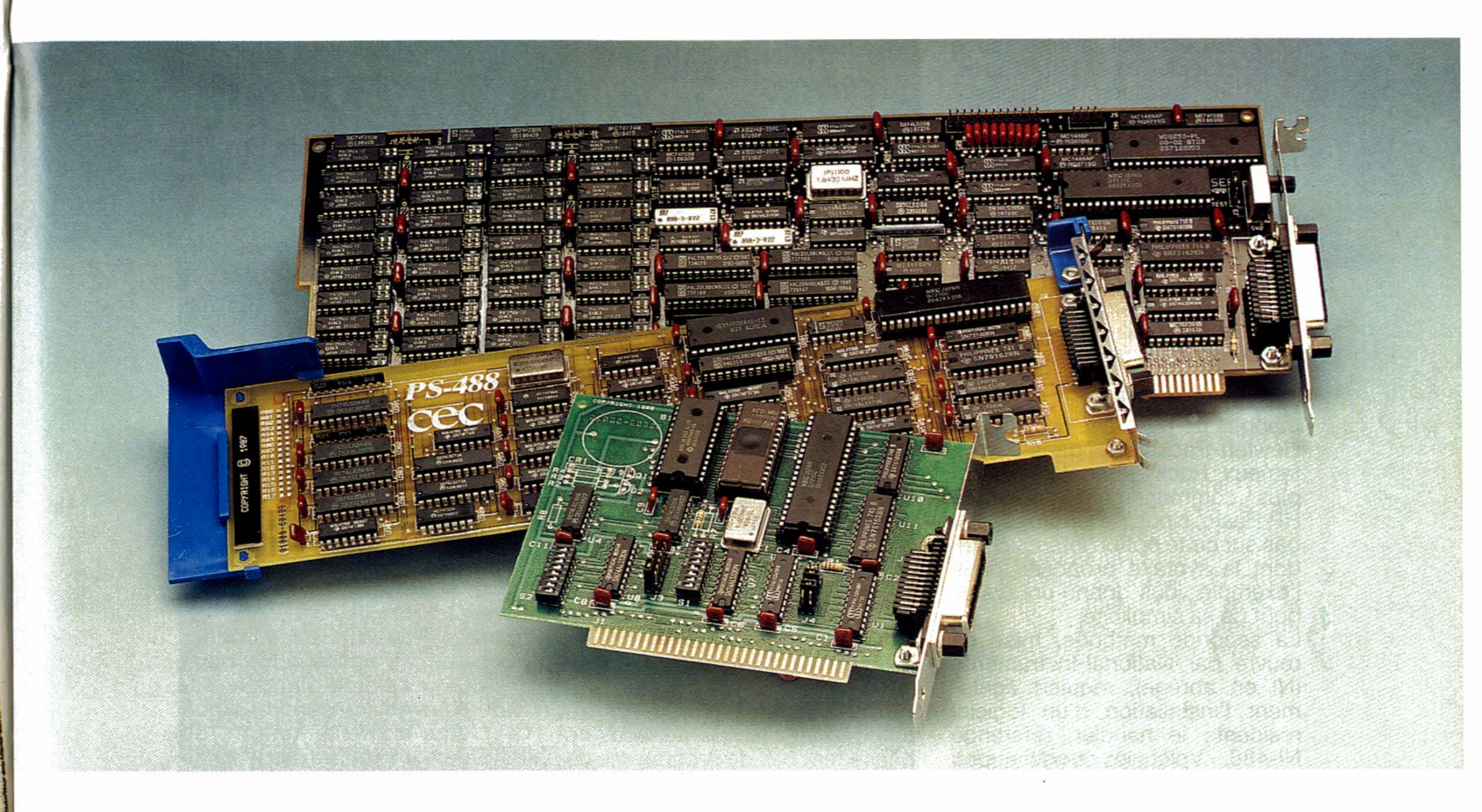
HT BASIC
LE BASIC HP SUR PC
NOUVEAU : COMPILATEUR



Les Ulis - Tél : 69 07 41 42

Sud-ouest - Tél : 61 39 15 16 - Provence - Tél : 42 60 00 42

La programmation des cartes IEEE-488 pour PC



Après l'article consacré à l'architecture du bus IEEE-488 paru dans le nº 534 de mai, la présente publication vous invite naturellement à découvrir les bases de la programmation des interfaces GPIB. A cet effet, les lignes suivantes décrivent en détail les principes fondamentaux nécessaires à la bonne utilisation du bus.

Bien qu'il existe de nombreuses cartes IEE disponibles dans le commerce, l'étude que nous proposons portera sur trois d'entre elles : la PCIIA de National-Instruments, la 82335A de Hewlett-Packard et enfin, la PC-488 de Capital Equipment Corporation, distribuée par Keithley France. Afin que le lecteur puisse directement lier nos propos avec une application pratique, nous commenterons plusieurs programmes écrits en langage C, immédiatement exploitables sur votre PC équipé de la carte IEEE citée. Ces courts logiciels sont destinés à accélérer la prise en main de l'interface, et certaines portions pourront intégrer un futur ensemble de test. Un court paragraphe introduira en fin d'article le logiciel HT Basic, destiné à recréer l'environnement de programmation HP Basic d'un HP 9000 sur un compatible PC.

INSTALLATION ET COMMUNICATION AVEC L'INTERFACE IEEE

Selon le constructeur, le contrôleur IEEE se présente sous la forme d'une carte aux dimensions compactes 8 ou 16 bits, comme en témoignent les diverses photographies. Le module prend place dans l'un des slots du compatible et occupe une adresse mémoire au sein de l'espace I/O autorisé. On vérifiera alors l'absence de tout conflit d'adresses si plusieurs cartes de natures différentes doivent coexister au moment de l'installation.

L'emploi des commandes IEEE à l'intérieur d'un programme, s'effectue suivant trois méthodes : la première consiste à ouvrir un fichier créé grâce à la présence d'un programme résident, au travers duquel on expé-

die des ordres IEEE :
FILE * board ; /*descripteur de l'interface IEEE*/.
board = fopen ("gpib", " + r") ; /*ouvre le descripteur*/.
fputs ("OUTPUT 1 ; VDC ; AUTO" board) ; /*envoie la chaîne dans le fichier*/.
En anglais, ce principe se nomme "character I/O driver", car le programme ainsi compilé envoie des chaînes de caractères dans un fichier tampon. Ces chaînes décrivent la commande GPIB associée à ses paramètres, là où les adresses du dispositif concerné, et si nécessaire, les ordres de programmation (VDC, AUTO...). Cette façon de procéder ralentit l'expédition des messages, qui nécessitent une analyse syntaxique complète afin d'être décodés puis finalement exécutés (que signifie OUTPUT 1 ?). De plus, le programme résident consomme légèrement la mémoire du compatible. Cependant, ce type de driver offre l'avantage d'accepter des commandes de bus développées par Hewlett-Packard dans son fameux Rocky Mountain Basic, plus communément dénommé HP Basic. Nous reviendrons plus bas sur cette option intéressante, dont la syntaxe proposée dans l'exemple vous donne un avant-goût de sa simplicité.
La seconde méthode, mise en œuvre par National-Instruments (NI en abrégé), requiert également l'installation d'un logiciel résident : le handler, référencé NI-488. Voici les commandes équivalentes à celles évoquées dans l'exemple précédent :
int descripteur ;
descripteur = ibfind (instrument) ; /*retourne le descripteur de l'instrument*/
ibwrt (descripteur, "VDC ; AUTO", 9) ; /*expédie 9 caractères à l'instrument*/
La différence fondamentale avec le précédent système, se trouve dans le style de programmation qui appelle directement des fonctions ou routines, immédiatement interfacées par ce handler. Plus besoin de traduction ou d'appels à des fichiers tampon, puisque chaque opération utilise une routine clairement définie, à laquelle on passe les paramètres par l'intermédiaire de la pile. La liaision directe avec l'interface transite ensuite via l'espace I/O du compatible. En anglais, vous retrouverez dans les notices, le terme "Subroutine Structured Driver" qui résume ce que nous venons d'énoncer. Le jeux d'instructions proposé par NI contient de nombreuses routines bas et haut niveau. Le constructeur laisse alors libre choix à l'utilisateur d'opter pour le style qui lui convient.
Enfin, la troisième méthode exploitée par C.E.C. et Hewlett-Packard, ne demande aucun driver résident pour attaquer leurs cartes IEEE. C.E.C., par exemple, construit des interfaces dont le logiciel se trouve en ROM. Les appels en Basic s'effectuent alors directement sur ce composant via l'espace mémoire du PC. En C, par contre, il suffit de lier le programme compilé avec la librairie fournie par les deux fabricants, pour attaquer la carte via les entrées/sorties de l'ordinateur. Puisque les interfaces ne possèdent pas de programme de configuration, les appels utilisent des adresses par défaut. En cas de modifications des interrupteurs de la carte, les constructeurs fournissent des routines nécessaires à la correcte initialisation du hardware. Nous reviendrons sur ces particularités au cours des descriptions détaillées proposées plus loin.



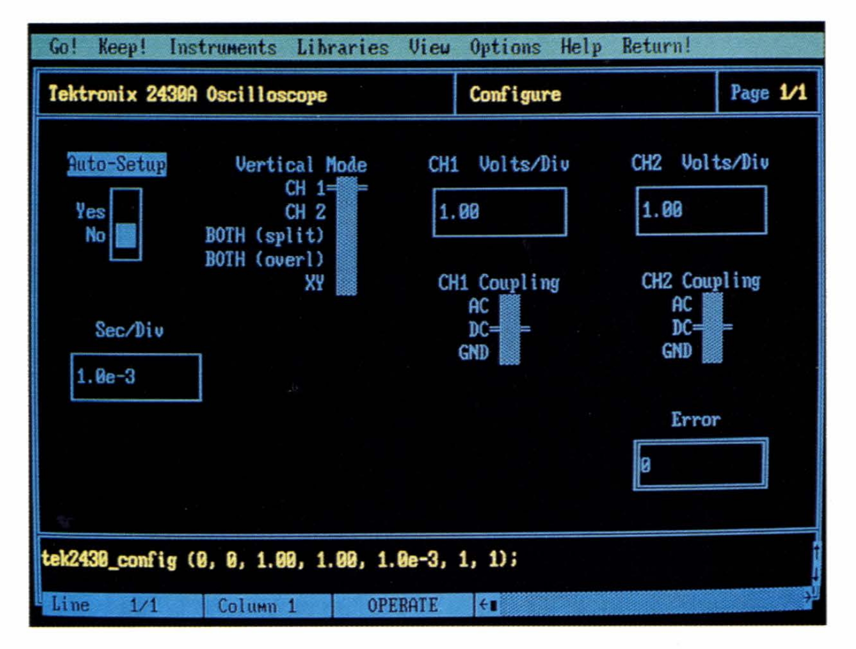
LA PROGRAMMATION DES CARTES NATIONAL-INSTRUMENTS

Nous utilisons la carte PCIIA, largement répandue dans les laboratoires. Elle n'occupe qu'un emplacement huit bits, et sa facile mise en œuvre la rend accessible à tous les gens qui développent du logiciel. Comme nous venons de le souligner, la carte NI réclame l'installation d'un handler (ou driver), le programme résident GPIB.COM. Il suffit d'inclure la ligne suivante dans le fichier config.sys pour qu'il soit automatiquement chargé au démarrage du PC :
device = chemin gpib.com. En principe, l'utilitaire qui gère l'installation de votre carte, se charge d'ajouter cette ligne dans ce fichier de configuration. Par la suite, le lancement du programme IBCONF.EXE permettra de déclarer puis configurer tous les équipements reliés au bus GPIB dont le contrôleur lui-même. Le handler pilote directement le module IEEE et nécessite l'ajout d'une interface langage afin de dialoguer correctement avec le compilateur utilisé (C, Pascal, Fortran, Basic...). Pour notre part, il s'agit du fichier objet TCIB.OBJ puisque l'auteur travaille sous environnement Turbo C. Cette nouvelle version de TCIB gère les quatre modèles de mémoire tiny, small, compact et large.
Si l'on développe sous l'environnement Borland, on ouvrira un fichier projet, comprenant les fichiers.C développés et TCIB.OBJ, qui sera alors automatiquement pris en compte lors de l'édition de liens. On signalera que cette interface langage **Turbo C** ne figure pas en standard sur les disquettes d'installation NI. Il vous faudra le préciser lors de votre commande.
La déclaration des nombreux prototypes de fonction, ainsi que les diverses constantes, figurent dans le fichier header DECL.H que l'on prendra soin d'inclure en début de fichier par la ligne :
include "decl.h".

Les fonctions du driver NI-488

Tout appel de fonction NI possède le format suivant :
ibfonction (ud, liste de paramètres) ;
ibfonction représente l'une des trente routines accessibles de l'interface langage, associées à

son inévitable liste de paramètres. Afin d'expédier les ordres au bon endroit, la communication avec un dispositif connecté sur le bus passe par un "unit descriptor" (ud) qui autorise des opérations d'entrée/sortie avec cet instrument. La première des choses à faire consiste donc à "ouvrir" un appareil IEEE via le handler, par la fonction ibfind qui retourne alors le descripteur correspondant :
ud = ibfind ("fluke 45") ; /*Ouvre le descripteur du multimètre Fluke 45*/
On notera cependant, qu'il faut travailler avec des noms d'instruments connus du handler, grâce au programme IBCONF.EXE. Si



l'appareil appelé ne figure pas à l'intérieur de la configuration, la fonction ibfind retourne alors une erreur.

Les variables d'environnement

Chaque appel à une fonction NI rafraîchit les variables entières ibsta, iberr et ibcnt. La première décrit l'état du bus et de sa carte interface associée, tandis que la seconde ne possède un sens qu'en présence d'une erreur, indiquée par le bit 15 de ibsta alors armé (bit nommé ERR). Nous insistons sur l'aspect indispensable de tester ibsta et, éventuellement iberr (un simple ET d'ibsta avec 8 000 hexa), à chaque requête au handler via une ibfonction. Ce test, qui peut revêtir la forme d'une simple ligne, permettra d'alerter l'utilisateur d'une erreur apparue sur le bus :
ibfonction (ud, param1, param2...) ; /*Appel à une fonction NI-488*/
if (ibsta & ERR) erreur – routine() ; /*Si le bit 15 est armé, gestion de l'erreur*/
ou encore, si l'on veut condenser le code :
if (ibfonction (ud, param1, param2...) & (ERR/TIMO)) erreur – routine() ;
Dans ce dernier cas, nous vérifions également l'absence de time-out, considéré comme une erreur de fonctionnement (l'appareil ne répond pas dans le délais imparti).
Par expérience, un programme de test qui commence à planter de façon imprévue autorise un déverminage nettement plus efficace lorsque tous les tests de variables existent, et renseignent le concepteur au travers de l'écran sur l'état du bus IEEE.

Ibcnt représente la variable compteur entière, qui indique à l'utilisateur le nombre d'octets transférés durant les opérations IEEE. Ibcnt se définit sur 16 bits, alors que sa collègue Ibcntl, absolument identique, exploite 32 bits (long integer).

Ecritures et lectures

L'envoi de caractères à un équipement, s'effectue via la fonction ibwrt à laquelle on passe la chaîne à transmettre, suivie du nombre total de caractères émis :
ibwrt (ud, "VDC", 3) ; /*Sélectionne VDC (3 carractères) sur l'équipement ud*/
if (ibsta & ERR) erreur – routine() ; /*Si le bit 15 est armé, gestion de l'erreur*/
NI-488 n'envoie uniquement que les caractères contenus entre les guillemets. Cette remarque concerne la fin d'écriture, éventuellement choisie avec un Line-Feed, que l'on devra obligatoirement inclure dans la chaîne expédiée si l'on souhaite la transmettre (voir fonction ibeos du manuel NI).
La lecture d'un résultat exploite la fonction ibrd, qui réclame simplement le nombre d'ocets à lire. Rappelons que la lecture s'achève lorsque l'instrument adressé en parleur valide la ligne EOI ou envoie le caractère de fin de transmission (si cette option est activée au sein du handler). On peut sélectionner différents caractères d'arrêt de lecture sous l'utilitaire IBCONF.EXE, ou bien à l'aide des fonctions ibeos et ibconf. Voici un exemple de programmation, illustrant l'utilisation d'ibrd :
char buffer [100] ; /*Stockage des caractères reçus*/
ibrd (ud, buffer, 50 ;) /*Range 50 caractères émis par ud, dans buffer*/
if (ibsta & ERR) erreur-routine () ; /*Si le bit 15 est armé, gestion de l'erreur*/
Il existe d'autres routines permettant la gestion des lignes IEEE, ainsi que des fonctions de niveaux supérieurs qui simplifient grandement le travail du développeur. Le manuel livré avec la carte les décrit parfaitement, avec de nombreux exemples.

Acquisition de données avec un multimètre Fluke 45

Plutôt que de décrire étape par étape l'écriture d'un logiciel d'acquisition, nous vous proposons de décortiquer sans tarder, le programme IEEE-488.C proposé en **figure 2**, qui pilote la configuration de la **figure 1**. Cet exemple réalise l'échantillonnage d'une tension à l'aide d'un Fluke 45, déclenché cent fois par le bus IEEE. Le stockage des données dans un fichier se transformera en courbe, grâce à un traitement approprié disponible sous un tableur tel que Excel, de MicroSoft.

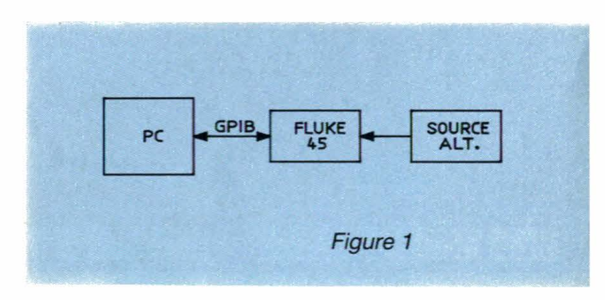


Figure 1

```
/*
    Programme d'acquisition et stockage de données
    lues par un multimètre FLUKE 45 selon NI-488.
    IEEE488.C, Christophe BASSO 1992
*/

#include <stdio.h>
#include <stdlib.h>

#include "decl.h"                          /* Prototypes des fonctions NI488 */
#include "bus488.h"               /* Prototypes des fonctions relatives au Fluke */
#include "ibicerr.h"             /* Prototypes des fonctions de gestion d'erreur */

int    Ud;                           /* Descripteur de l'instrument adressé */
FILE *dat_file;                          /* Pointeur du fichier de données */

#define path_file "d:\\tc\\bus488\\fluk_acq.dat"

void main(void)
{
    int loop;
    float valeur;

/* -------------------- Initialisation du Fluke ---------------------------*/

    init_fluke();

/* -------------------- Ouverture du fichier de données -------------------*/

    if ((dat_file = fopen(path_file,"wt")) == NULL )
          fail("Unable to open destination file\a");

/* -------------------- Boucle de lecture/stockage de 100 données ---------*/

    for (loop=0;loop < 100;loop++)
    {
        valeur = read_fluke();
        printf("Fluke45 = % f\n",valeur);
        fprintf(dat_file,"%d\t% f\n",loop,valeur);
    }

/* ------------------- Termine le programme proprement ------------------- */

    cleanup();
}
```

Figure 2

```
/*
    Gestion du fluke selon la norme IEEE 488.1
    BUS488_1.C, Christophe BASSO
*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include "decl.h"                          /* Prototypes des fonctions NI488 */
#include "bus488.h"               /* Prototypes des fonctions relatives au Fluke */
#include "ibicerr.h"             /* Prototypes des fonctions de gestion d'erreur */

void cleanup(void)
{
    ibwrt(Ud,"*RST",4);              /* Remet le Fluke dans son état initial */
    show_err("Ibwrt");
    ibloc(Ud);                                /* Repasse le Fluke en local */
    show_err("Ibloc");
    ibonl(Ud,0);                     /* Referme le descripteur de l'appareil */
    show_err("Ibonl");
    if (fclose(dat_file) == EOF) fail("Unable to close destination file\a");
    puts("Program terminated");
}

void init_fluke(void)
{
    char *buf1;

    if ((Ud = ibfind("fluke45")) & ERR) show_err("Ibfind");
    else {
            buf1 = "*RST; VDC; AUTO; TRIGGER 2; *SRE 16";
            ibwrt(Ud,buf1,strlen(buf1) );
            show_err("Ibwrt");
         }
/*
    *RST        = initialisation du Fluke45
    VDC         = gamme tension continue
    AUTO        = selection automatique des échelles
    TRIGGER 2 = déclenchement externe par le bus
    *SRE 16     = déclenchement d'un SRQ en cas de donnée prête
    VAL1?       = retourne la valeur de l'affichage primaire
*/
}

float read_fluke(void)
{
    char fluke_state;                               /* mot d'état du Fluke45 */
    char buffer[25];                     /* buffer de stockage de la donnée lue */

/* ----------- Déclenche le multimètre ---------------------------------- */

    ibtrg(Ud);
    show_err("Ibtrg");

/* ----------- Demande la valeur lue ------------------------------------ */

    ibwrt(Ud,"VAL1?",5L);
    show_err("Ibwrt");

/* ----------- Attend l'arrivée du SRQ ou d'une erreur ------------------ */

    ibwait (Ud,TIMO|RQS);
    show_err("Ibwait");

/* ----------- Le SRQ présent, effectue un Serial Polling --------------- */

    ibrsp(Ud,&fluke_state);
    show_err("Ibrsp");

/* ----------- Vérifie que le statut du Fluke n'indique pas d'erreur ----- */

    if (fluke_state != 0x50) fail("Fluke 45 Error\a");

/* ----------- Lit la valeur présente dans le buffer du Fluke ------------ */

    ibrd(Ud,buffer,20L);
    show_err("Ibrd");

/* ----------- Retourne la valeur convertie en float -------------------- */

    return((float)atof(buffer));
}
```

Figure 3

LE LOGICIEL COMPLET

Il s'articule autour de trois sources, IEEE488.C, BUS488–1.C et IBICERR.C. Le premier comprend le mot clé "main" et correspond donc au programme principal. Le second, proposé en **figure 3**, permet le pilotage du Fluke 45 en l'initialisant puis en récupérant ses données. Le dernier, IBICERR.C en **figure 4**, gère les éventuelles erreurs apparues lors des transactions IEEE et alerte l'utilisateur par des messages appropriés. Le fichier projet (extension .PRJ) inclura donc les trois fichiers cités ainsi que l'interface langage TCIB.OBJ.

IEEE488.C

En début de programme, on retrouve les classiques fichiers d'en-tête (header files, en anglais) qui contiennent les prototypes, et constantes diverses, de toutes les routines appelées dans notre source. Lors de la compilation, Turbo C peut vous signaler l'absence de prototypes des fonctions utilisées si le fichier .H s'y rapportant n'est pas inclus (prototype not found). Il s'agit, dans notre cas, de STDIO.H et STDLIB.H qui déclarent, entre autres, des fonctions telles printf ou encore fopen, en précisant au compilateur les variables que ces routines retournent ou encore, les divers arguments à leur passer. Au cas où votre appel comprendrait des éléments de type incompatible avec la déclaration (passer un entier à la place d'un décimal, par exemple), le compilateur produirait alors une erreur du type "type mismatch in parameter xd" vous indiquant l'incompatibilité de l'appel avec la déclaration du prototype sur le xième paramètre. Afin d'éviter de tels tracas, on prendra soin d'inclure tous les fichiers d'en-tête qui décrivent les prototypes des routines que nous avons écrites dans les sources, ou qui figurent dans une librairie qui sera liée ultérieurement. Il s'agit pour notre application, des programmes IBICERR.C et BUS488–1.C auxquels correspondent respectivement les fichiers de déclarations IBICERR.H et BUS488–1.H, disponibles en **figure 5**.

La présence des guillemets à la place des signes inférieurs et supérieurs, indiquent au compilateur que ces fichiers existent dans le répertoire courant et non dans le répertoire INCLUDE, qui comprend par défaut les headers des librairies classiques de Turbo C. DECL.H, quant à lui, contient

```c
/*
   Traitement des erreurs apparues sous GPIB
   IBICERR.C, Christophe BASSO 1992
*/

#include <stdio.h>
#include <stdlib.h>

#include "decl.h"                 /* Prototypes des fonctions NI488 */
#include "ibicerr.h"        /* Prototypes des fonctions de gestion d'erreur */

void fail(char *s)
{
    puts(s);                               /* Affiche le message d'erreur */
    puts("Program aborted");
    exit(EXIT_FAILURE);
}

void show_err(char *message)
{
    if (ibsta & (ERR | TIMO))               /* Erreur GPIB ou time-out ? */
    {
       printf("Error when executing <%s>\n\a",message);
       ibstatus(ibsta);
       iberror(iberr);
       puts("Program aborted");
       exit(EXIT_FAILURE);
    }
}

void iberror(int erreur)
{
    switch(erreur)
    {
       case EDVR : puts("DOS error");break;
       case ECIC : puts("Function Requires GPIB board to be CIC");break;
       case ENOL : puts("Handshake error (e.g., no listener)");break;
       case EADR : puts("GPIB board not addressed correctly");break;
       case EARG : puts("Invalid argument to function call");break;
       case ESAC : puts("GPIB board not system controller as required");break;
       case EABO : puts("I/O operation aborted (time out)");break;
       case ENEB : puts("Non-existent GPIB board");break;
       case EOIP : puts("Asynchronous I/O in progress");break;
       case ECAP : puts("No capability for operation");break;
       case EFSO : puts("File system error");break;
       case EBUS : puts("GPIB bus error");break;
       case ESTB : puts("Serial poll status byte queue overflow");break;
       case ESRQ : puts("SRQ stuck in ON position");break;
       case ETAB : puts("Table problem");
    }
}

void ibstatus(int status)
{
    if (status & DCAS) puts("Device clear State");
    if (status & DTAS) puts("Device Trigger State");
    if (status & LACS) puts("Listener");
    if (status & TACS) puts("Talker");
    if (status & ATN)  puts("Attention is asserted");
    if (status & CIC)  puts("Controller-In-charge");
    if (status & REM)  puts("Remote State");
    if (status & LOK)  puts("Lockout State");
    if (status & CMPL) puts("I/O completed");
    if (status & RQS)  puts("Device requesting service");
    if (status & SRQI) puts("SRQ interrupt received");
    if (status & END)  puts("END or EOS detected");
    if (status & TIMO) puts("Time limit exceed");
    if (status & ERR)  puts("GPIB error");
}
```

Figure 4

```c
/*
   IBICERR.H
   Déclaration des prototypes contenus dans ibicerr.c
*/

#ifndef __ibic
#define __ibic

void show_err(char *);                  /* Traitement de la variable ibsta */
void iberror(int);                      /* Affichage de la variable iberr */
void ibstatus(int);                     /* Affichage de la variable ibsta */
void fail(char *);                      /* Affichage d'un message d'erreur */

#endif

/*
   BUS488.H
   Déclaration des protypes et variables des fichiers
   IEEE488_1.C et IEEE488_2.C
*/

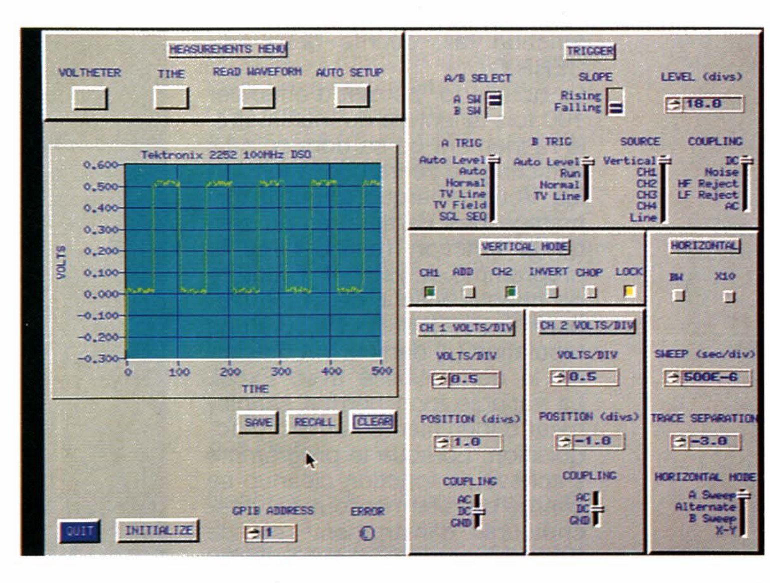
#ifndef __488
#define __488

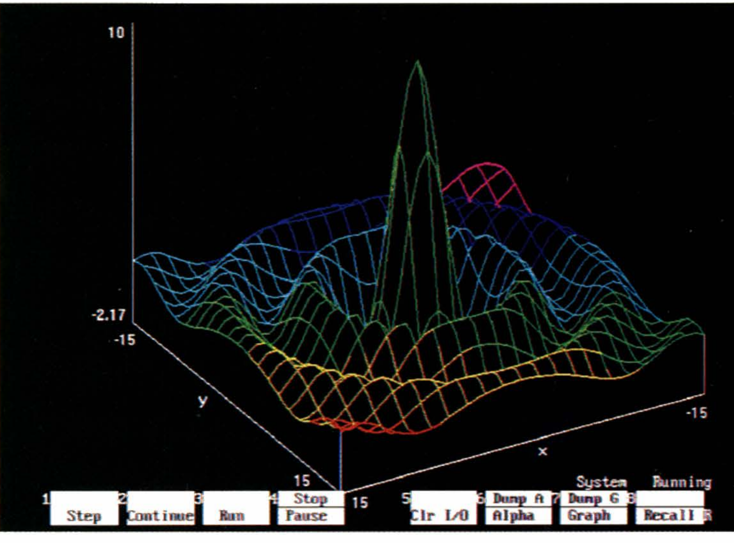
extern int   Ud;
extern FILE *dat_file;

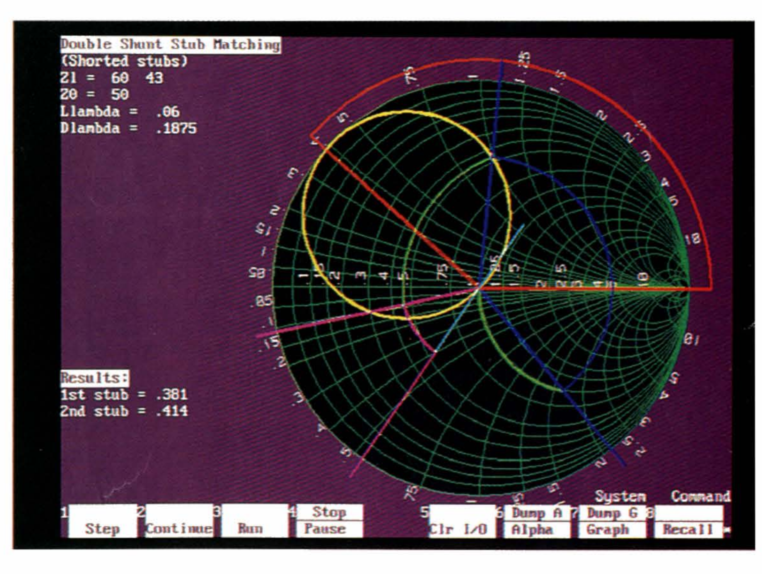
void  cleanup(void);                  /* Termine proprement le programme */
void  init_fluke(void);             /* Initialise le multimètre Fluke 45 */
float read_fluke(void);        /* Retourne la valeur convertie par le Fluke */

#endif
```

Figure 5

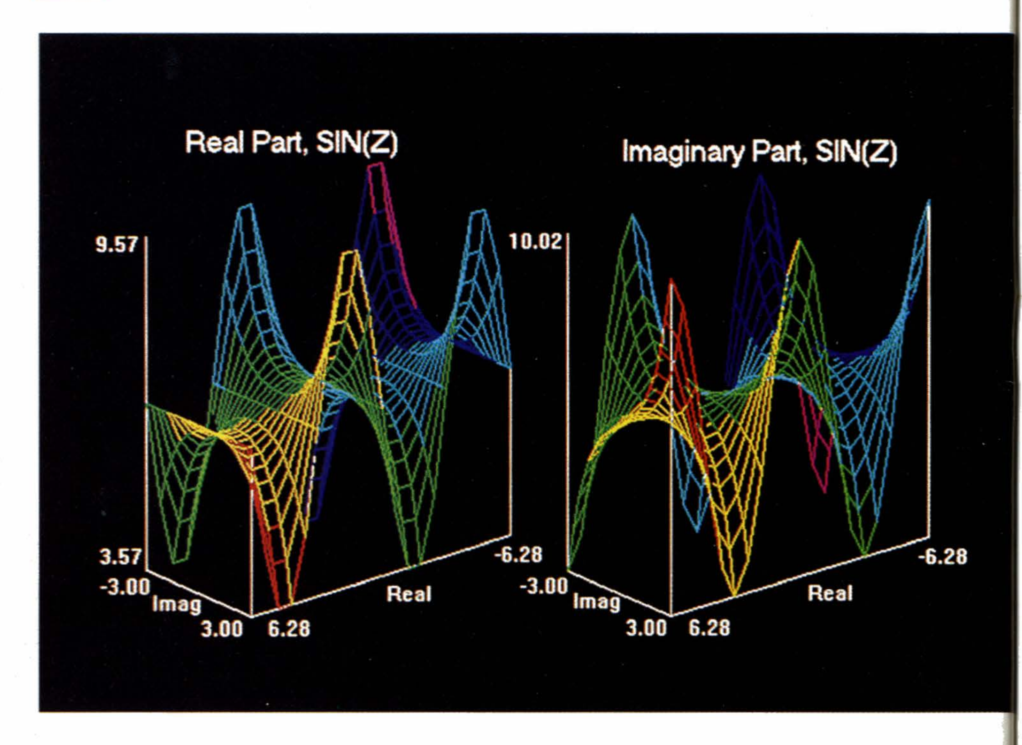






toutes les déclarations et nombreuses constantes des routines offertes par la librairie TCIB.OBJ. Nous n'allons pas nous appesantir sur les variables globales qui définissent le descripteur d'instrument ainsi que le pointeur du fichier de stockage, dont la ligne path–file définit plus bas son chemin d'accès. Attaquons à présent le début du programme principal...
Avant toute opération, il convient d'initialiser le Fluke afin qu'il fonctionne selon nos désirs. La fonction init–fluke se charge de cette opération dans le programme BUS488–1.C sur lequel nous reviendrons dans quelques instants. La ligne suivante ouvre le fichier de stockage en mode écriture afin d'y ranger chaque résultat d'acquisition émis par le Fluke. Si le pointeur retourne NULL, une erreur s'est produite durant l'ouverture et entraîne l'arrêt du programme grâce à la

fonction fail, décrite dans IBICERR.C.
La boucle for permet d'effectuer 100 fois la lecture du multimètre, le rangement de sa donnée puis l'affichage à l'écran de la tension lue. Vous remarquerez que le formatage des paramètres au sein de la fonction fprintf, force le programme à justifier l'écriture de chaque valeur décimale (espace entre % et f) et ajoute une tabulation (t) dans le but d'assurer la compatibilité avec Excel. La variable loop permet simplement d'identifier le numéro d'acquisition. Lorsque le programme s'achève, la fonction cleanup se charge de refermer tous les descripteurs d'instrument et de fichier.



BUS488-1.C

Le début du source contient, comme énoncé plus haut, les fichiers .H décrivant les prototypes des fonctions ou certaines variables utilisées dans ce programme. Commençons la description par init–fluke, puisqu'elle figure en premier dans le programme principal.
Avant tout appel au handler NI-488, il convient d'ouvrir le descripteur de l'instrument que l'on souhaite utiliser. La fonction IBFIND permet cette opération en retournant notre descripteur sous forme d'entier. Le test d'ibsta se trouve juste après et autorise la détection d'une éventuelle erreur d'ouverture : dispositif absent du handler, handler non chargé dans la mémoire... Par contre, puisqu'il s'agit exclusivement d'une opération soft, aucune erreur ne se manifeste si l'appareil n'existe pas physiquement sur le bus. Le test des variables ibsta et iberr prend forme dans la routine show–err, mise en œuvre dans IBICERR.C.
L'initialisation proprement dite, consiste à envoyer des ordres au multimètre, qui définiront sa configuration de travail. On se reportera à la documentation du Fluke ou de tout autre multimètre, pour expédier les commandes adéquates. Les commentaires présents dans le programme décrivent chaque ordre séparément.
Un mot cependant sur *SRE16, qui force l'instrument à déclencher un service request à la fin de sa conversion, indiquant la disponibilité de sa donnée. La valeur 16 masque le registre de programmation du multimètre conformément aux informations fournies par le constructeur. La seule remarque concerne le passage de la chaîne complète à la fonction ibwrt, par l'intermédiaire d'un pointeur de caractères qui, grâce à la fonction strlen, nous affranchit du décomptage exact des caractères envoyés sur le bus.
La lecture du Fluke 45 réclamera au préalable pour notre exemple, un déclenchement externe par le bus tel que nous l'avons demandé par l'intermédiaire de la chaîne TRIGGER 2 : la routine ibtrg accomplit cette fonction. Une fois déclenché, le multimètre ne fournit la valeur de son affichage primaire, qu'à réception de l'ordre VAL1 ?. A partir de cet instant, l'instrument doit avertir le contrôleur par le biais de la ligne SRQ qu'il réclame son attention. Soit parce que la mesure est prête (ce que nous lui avons ordonné lors de son initialisation) ou bien qu'une erreur existe (conversion erronée, décalibration...). Ainsi, avant d'effectivement récupérer par ibrd le résultat de l'acquisition, il faut attendre l'arrivée du SRQ : la fonction ibwait se charge alors de tester la validation éventuelle du bit SRQI géré par l'interface PCIIA. Une fois le service request apparu, il convient d'interroger l'équipement sur la raison de sa demande d'attention.
Le polling série sur l'instrument considéré, mené par ibrsp, nous renvoie alors son mot d'état sous la forme d'un simple octet. Si ce mot diffère avec celui que l'on attend (mesure disponible, soit 0x50 d'après la documentation), une routine de test s'occupera de prendre les dispositions qui s'imposent. Dans notre cas, le programme s'interrompt en affichant un message d'erreur. Le lecteur notera que le contrôleur IEEE procède automatiquement à chaque arrivée de SRQ, au polling série de l'instrument qui réclame l'atttention : il s'agit de l'option "Enable Auto Serial Polling" au sein d'ibconf. Le mot d'état récupéré est alors stocké, puis délivré lorsque le programme appelle la fonction ibrsp.
A présent la donnée existe et le Fluke attend que le contrôleur vienne la récupérer à l'aide de ibrd. Nous demandons la lecture d'une vingtaine de caractères, sachant que le Fluke n'en utilise jamais plus pour convertir la valeur mesurée. La dernière ligne transforme la chaîne en valeur décimale, que la sous-routine renvoie finalement au programme principal qui l'appelait.

Remarque :

Il existe une autre méthode de lecture généralement employée par les débutants, qui consiste à déclencher le multimètre, puis après une attente (delay ou sleep), lire la valeur mesurée. L'inconvénient majeur de ce style de programmation réside dans l'impossibilité de tester l'état de l'instrument après chaque conversion. La valeur retournée peut être erronée, rien n'avertit l'utilisateur. De plus, le délai nécessaire à la complète conversion dépend du temps de calcul propre à l'équipement qui peut, dans certains cas, varier de quel-

ques centaines de ms à plusieurs secondes. Si ce délai possède une trop faible valeur, le contrôleur ne lit rien puisqu'aucune donnée n'est disponible.

Dans le cas où l'équipement ne supporte pas le service request, la scrutation par polling série de son mot d'état jusqu'à ce que le bit "données prêtes" soit armé, constitue une alternative possible et efficace. Nous l'aborderons un peu plus loin.

La fonction cleanup ferme tous les descripteurs ouverts au lancement du programme. Pour ceux retournés par le handler, Ibonl se charge de les refermer individuellement. Au cas où cette commande ne figurerait pas dans le source, le lancement du même programme plusieurs fois de suite conduirait à une erreur système, puisque le DOS limite le nombre maximum de fichiers ouverts (FILE = ,dans config.sys).

IBICERR.C

Ce programme teste à chaque retour d'une ibfonction la présence éventuelle d'une erreur GPIB ou d'un time-out. Ibsta et iberr sont alors retournées sous forme d'un entier de deux octets, dont chacun des bits revêt une signification. En cas de problème, un ou plusieurs messages avertissent l'utilisateur de l'activité sur le bus. On se reportera à la documentation N.I pour plus de renseignements au sujet des variables ibsta et iberr.

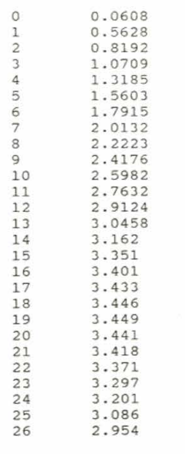
Résultat des acquisitions

Un extrait du fichier vous est proposé en **figure 6**. Excel traite ces données et fournit toutes sortes de graphiques, selon les goûts et besoins de chacun. Celui de la **figure 7**, correspond au tracé de la sinusoïde présente à l'entrée du Fluke lors des acquisitions. La distorsion à certains endroits, provient d'une part du générateur à cette basse fréquence, puis du changement de gamme automatique du Fluke qui retarde la prise d'échantillon lors de ses commutations internes.

Les applications de ce type de programme sont très nombreuses. Elles concernent la prise de données notamment sur des périodes longues, comme par exemple les variations de courant sur un panneau solaire durant 24 heures...

Les interruptions sous NI-488

Malheureusement, la carte PCIIA ne supporte pas, avec les drivers livrés **en standard**, les vrais appels d'interruptions en langage C déclenchés par un SRQ. Cette particularité aurait permis l'écriture optimisée d'un programme en évitant une boucle d'attente jusqu'à l'apparition d'un Service Request de l'instrument concerné (fonction ibwait).



0	0.0608
1	0.5628
2	0.8192
3	1.0709
4	1.3185
5	1.5603
6	1.7915
7	2.0132
8	2.2223
9	2.4176
10	2.5982
11	2.7632
12	2.9124
13	3.0458
14	3.162
15	3.351
16	3.401
17	3.433
18	3.446
19	3.449
20	3.441
21	3.418
22	3.371
23	3.297
24	3.201
25	3.086
26	2.954

Figure 6

Cependant, National-Instruments propose la fonction ibsrq qui installe un gestionnaire d'interruption, appelé lorsque le bit SRQI de ibsta se trouve armé. Il nécessite un rafraîchissement d'ibsta par des appels périodiques au handler et, en ce sens, ne constitue pas de véritable gestion d'interruption. La ligne ibwait (ud, 0), permet de renouveler la valeur d'ibsta périodiquement dans le logiciel.

L'utilisation des commandes bas niveau

National-Instruments autorise, grâce à des fonctions bas niveau, l'expédition de commandes multilignes, unilignes et adressées sur le bus GPIB. Par exemple, valider la ligne Remote Enable (REN) s'écrit : ibsre (ud, 1) ou encore, la ligne IFC se pilote par : ibsic (ud). Les commandes multilignes adressées, utilisent la fonction ibcmd qui permet d'envoyer des ordres tels que, entre autres, GET, MLA, MTA, SPD... Ceux d'entre vous qui développent des interfaces IEEE pour des instruments apprécieront la présence de telles routines.

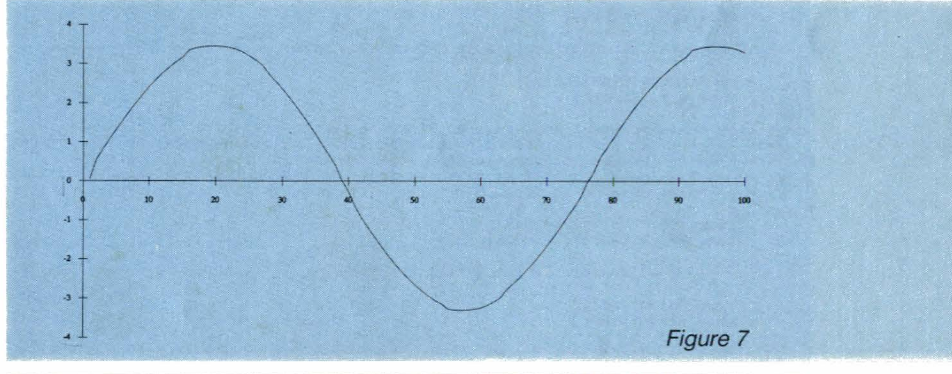
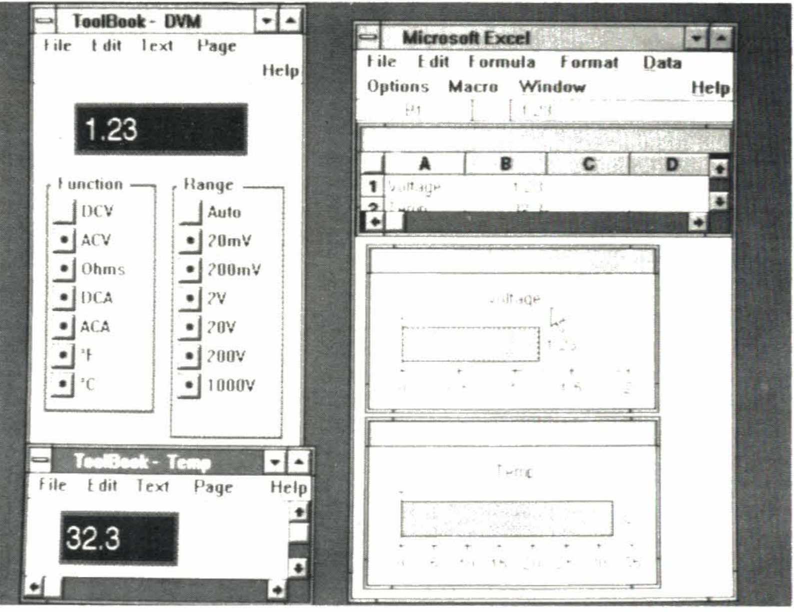


Figure 7



```c
/*
   Exemple d'utilisation des fonctions bas niveau IEEE-488
   fichier BOARDO.C, Christophe BASSO 1992
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "decl.h"                      /* Prototypes des fonctions NI488 */
#include "ibicerr.h"      /* Prototypes des fonctions de gestion d'erreur */

void main(void)
{
   char  *buf1,
         buffer[25];          /* Stockage des caractères émis par le Fluke */
   int   Ud,                      /* Descripteur de l'instrument adressé */
         loop;                          /* Nombre de mesures affichées */

/* ---------- Ouvre le descripteur du controleur GPIB0 ------------------ */

   if (( Ud = ibfind("GPIB0")) < 0) fail("Device Ibfind error ...");

/* ---------- Reset le GPIB0 par la commande uniligne IFC ---------------- */

   ibsic(Ud);
   show_err("Ibsic");

/* ---------- Valide la ligne REN par la commande uniligne ibsre --------- */

   ibsre(Ud,1);
   show_err("Ibsre");

/* ---------- Inhibe le mode local du Fluke 45 (LLO)  : 0x11 ------------ */
/* ---------- Place le Fluke en remote en l'adressant : 0x25 ------------ */
/* ---------- Reset le Fluke 45 par la commande DCL   : 0x14 ------------ */
/* ---------- Place le GPIB en mode Talker            : 0x40 ------------ */

   ibcmd(Ud,"\x11\x25\x14\x40",4L);
   show_err("Ibcmd");

/* ---------- Initialise le Fluke 45 ------------------------------------- */

   buf1 = "*RST; VDC; AUTO; TRIGGER 2; *SRE 16";
   ibwrt(Ud,buf1,strlen(buf1) );
   show_err("Ibwrt");

   for (loop=0;loop < 10;loop++)
   {

/* ---------- Demande au Fluke 45 d'écouter (MLA5) : 0x25 ---------------- */
/* ---------- Passe le GPIB0 en mode parleur       : 0x40 ---------------- */

     ibcmd(Ud,"\x25\x40",2L);
     show_err("Ibcmd");

/* ---------- Déclenche le Fluke par la commande GET : 0x08 -------------- */

     ibcmd(Ud,"\x08",1L);
     show_err("Ibcmd");

/* ---------- Demande au Fluke de délivrer sa mesure -------------------- */

     ibwrt(Ud,"VAL1?",5L);

     show_err("Ibwrt");

/* ----------- Attend l'arrivée du SRQ ou d'un time-out ----------------- */

     ibwait (Ud,TIMO|SRQI);
     show_err("Ibwait");

/* ---------- Stoppe l'adressage du Fluke 45 par UnTalk (UNT)    : 0x5F --- */
/* ---------- et lui demande de ne plus écouter, Unlisten (UNL) : 0x3F --- */
/* ---------- Envoie la commande de Serial Polling (SPE)         : 0x18 --- */
/* ---------- pour laquelle le Fluke 45 émet son statut (MTA5)   : 0x45 --- */
/* ---------- Demande au GPIB0 d'écouter sa réponse (MLA0)       : 0x20 --- */

     ibcmd(Ud,"\x5F\x3F\x18\x45\x20",5L);
     show_err("Ibcmd");

/* ----------- Récupère le mot d'état du Fluke 45 ---------------------- */

     ibrd(Ud,buffer,1L);
     show_err("Ibrd");

/* ---------- Teste la présence éventuelle d'une erreur ----------------- */

     if (buffer[0] != 0x50) fail("Fluke 45 Error");

/* ---------- Termine le Serial Poll en le dévalidant (SPD)      : 0x19 --- */

     ibcmd(Ud,"\x19",1L);
     show_err("Ibcmd");

/* ----------- Lit la valeur présente dans le buffer du Fluke ----------- */

     ibrd(Ud,buffer,20L);
     show_err("Ibrd");

/* ---------- Affiche le résultat de la mesure -------------------------- */

     printf("Fluke45 = % f\r\n",atof(buffer));
   }

/* ---------- Demande au Fluke 45 d'écouter (MLA5)               : 0x25 --- */
/* ---------- Passe le GPIB0 en mode parleur                     : 0x40 --- */
/* ---------- Réinitialise le Fluke par le message (*RST) ---------------- */
/* ---------- Repasse le Fluke en mode local (GTL)               : 0x01 --- */

   ibcmd(Ud,"\x25\x40",2L);
   show_err("Ibcmd");
   ibwrt(Ud,"*RST",4L);
   show_err("Ibwrt");
   ibcmd(Ud,"\x01",1L);
   show_err("Ibcmd");

/* ---------- Dévalide le GPIB0 ------------------------------------------ */

   ibonl(Ud,0);
   show_err("Ibonl");

   puts("Program terminated");
}
```

Figure 8

Pour les lecteurs qui souhaitent mettre en œuvre de telles écritures (bon courage...), le programme de la **figure 8**, BOARD0.C, vous permettra de vous familiariser avec ces fonctions bas niveau. Les commentaires ajoutés décrivent la chronologie du protocole retenu. On notera que toutes les commandes transitent via le GPIB0.

ULI ou la programmation en style Hewlett-Packard

Pour les inconditionnels des commandes mises en œuvre dans HP-Basic, le programme résident ULI devrait satisfaire leur passion. Cet utilitaire se charge par la commande ULI.COM à l'invite du DOS, et installe un "character I/O driver", comme nous l'avons décrit précédemment. Le programme proposé apparaît en **figure 9**, ULI–DEMO.C. Tous les appels transitent via le fichier gpib0, ouvert dès la première ligne. Il suffit ensuite d'expédier la commande sous forme d'une chaîne dans le fichier, pour que la fonction considérée soit ensuite présentée sur le bus. La séquence des commandes, décrite parles commentaires, se rapproche de celle développée dans la fonction read–fluke, décrite un peu plus haut. Les variables ibsta et iberr sont fournies sous forme de caractères individuels, impliquant leur conversion en entier avant de les tester (fonction check–bus). On notera que ULI contient déjà son propre gestionnaire d'iberr qui affiche un message d'erreur en cas de problème : la routine check–bus peut éventuellement être omise.

On remarquera la boucle sur le polling série, qui permet, en l'absence de test SRQ, d'attendre la fin de conversion du multimètre avant d'aller récupérer la donnée (bit 4 du mot d'état, armé).

Remarque :

ULI remplace l'interface langage habituelle et permet des appels par **n'importe quel type de compilateur**, à partir du moment où il sait expédier des caractères dans un fichier.
Cependant, vérifiez bien qu'ULI n'est pas installé (commande mem/debug à partir de DOS 4) avant de lancer l'un des programmes proposés plus haut lié avec TCIB.OBJ. Sinon, Crtl-Alt-Del... !

```
/*
   Exemple d'utilisation de l'Universal Language Interface ULI
   programme uli_demo.c, Christophe BASSO 1992
*/

#include <stdio.h>
#include <stdlib.h>

void ibstatus(int);
void fail(char *s) {
                        puts(s);
                        exit(EXIT_FAILURE);
                     }

void check_bus(FILE *bd)
{
   char ibsta_[5],
        iberr_[5],
        ibcnt_[5],
        header;
   int  ibsta;

/* ---------- Récupère les variables placées en tete de fichier --------- */

   if (fputs("STATUS\n",bd) == EOF) fail("Unable to perform STATUS");
   rewind(bd);
   if (fscanf(bd,"%c %5c,%5c,%5c",&header,ibsta_,iberr_,ibcnt_) == EOF)
       fail("No status byte read");

/* ---------- Affichage de la variable ibsta si erreur ------------------ */

   ibsta = atoi(ibsta_);
   if (ibsta & (0x8000 | 0x4000))    /* Erreur GPIB ou time out */
     {
        ibstatus(ibsta);
        exit(EXIT_FAILURE);
     }
   rewind(bd);
}

void ibstatus(int status)
{
   if (status & 0x1)     puts("Device clear State");
   if (status & 0x2)     puts("Device Trigger State");
   if (status & 0x4)     puts("Listener");
   if (status & 0x8)     puts("Talker");
   if (status & 0x10)    puts("Attention is asserted");
   if (status & 0x20)    puts("Controller-In-charge");
   if (status & 0x40)    puts("Remote State");
   if (status & 0x80)    puts("Lockout State");
   if (status & 0x100)   puts("I/O completed");
   if (status & 0x800)   puts("Device requesting service");
   if (status & 0x1000)  puts("SRQ interrupt received");
   if (status & 0x2000)  puts("END or EOS detected");
   if (status & 0x4000)  puts("Time limit exceed");
   if (status & 0x8000)  puts("GPIB error");
}

int get_status_byte(FILE *bd)
{
   char buf[3];

   if (fputs("SPOLL 5\n",bd) == EOF) fail("Unable to perform SPOLL");
   rewind(bd);

   if (fscanf(bd,"%3c",buf) == EOF) puts("Unable to get status byte");
   rewind(bd);
   return(atoi(buf));
}

void main(void)
{
   FILE  *bd;
   char  buf[25];
   int   loop;

/* ---------- Ouvre le descripteur du GPIB 0 ---------------------------- */

   if ( (bd = fopen("gpib0","w+")) == NULL) fail("Unable to open Board 0");

/* ---------- Initialise et reset le Bus -------------------------------- */

   if (fputs("ABORT\n",bd) == EOF) fail("Unable to perform ABORT");
   check_bus(bd);
   if (fputs("RESET\n",bd) == EOF) fail("Unable to perform RESET");
   check_bus(bd);

/* ---------- Programme le volt-mètre Fluke 45, adresse 5 --------------- */

   if (fputs("OUTPUT 5; *RST; VDC; AUTO; TRIGGER 2\n",bd) == EOF)
       fail("Unable to output device initialization sequence");
   check_bus(bd);

   for(loop=0;loop < 10;loop++)                      /* Effectue 10 mesures */
   {

/* ---------- Déclenche la conversion et demande la valeur -------------- */

      if (fputs("TRIGGER 5\n",bd) == EOF) fail("Unable to perform TRIGGER");
      check_bus(bd);

      if (fputs("OUTPUT 5; VAL1?\n",bd) == EOF)
      fail("Unable to perform VAL1?");
      check_bus(bd);

/* ---------- Attend la disponibilité de la mesure (bit 4 armé) --------- */

      while (!(get_status_byte(bd) & 0x10));

/* ---------- Effectue ensuite la lecture du volt-mètre ----------------- */

      if (fputs("ENTER 5#20\n",bd) == EOF) fail("Unable to perform ENTER");
      rewind(bd);
      if (fgets(buf,20,bd) == NULL) puts("No byte read");
      else printf("Fluke 45 = % f\n\r",atof(buf));
      rewind(bd);
   }

/* ---------- Passe en mode normal et libère le multimètre -------------- */

      if (fputs("OUTPUT 5; *RST\n",bd) == EOF)
          fail("Unable to perform *RST");
      check_bus(bd);

      if (fputs("LOCAL 5\n",bd) == EOF) fail("Unable to perform LOCAL");
      check_bus(bd);

      puts("Program terminated");
}
```

Figure 9

PROGRAMMATION DE LA CARTE HEWLETT-PACKARD

Nous utilisons la carte HP82335A, dont la photographie détaille l'implantation. Elle utilise le chip Texas TMS9914A, largement répandu dans le monde des contrôleurs IEEE. Comme annoncé prédémment, HP n'utilise pas de programme résident mais interface la carte directement via les entrées-sorties du PC, grâce à la librairie CLHPIB.

LIB fournie par le constructeur. Une remarque immédiate si vous compilez sous Turbo C (modèle de mémoire large) en utilisant un fichier projet : il faut absolument renommer cette librairie (sous DOS, renCLHPIB.LIB LHPIB.LIB, par exemple) car sans cela, Turbo C mélange ses crayons avec sa librairie CL.LIB et produit de nombreuses erreurs (Borland demande de ne pas utiliser de librairie dont le nom débute par CL...). C'est donc cette nouvelle appellation que l'on introduira dans le fichier projet. Si vous utilisez la compilation en ligne, le problème ne se pose pas : tcc-ml-L/lib chemin hp.c CLHPIB.LIB.

Initialisation de la carte

D'une façon similaire à NI qui spécifie le GPIB0 pour son interface IEEE, HP définit un code de sélection (ISC, Interface Select Code) qui identifie sa carte contrôleur. En général, il s'agit de la valeur 7, positionnée en usine. Cette particularité autorise la connexion de nombres cartes dans le PC, en leur attribuant des codes de sélection différents (voir documentation constructeur). A ce code correspond une adresse dans l'espace mémoire du PC. Pour la valeur 7, la carte s'installe au segment 0xDC00. En conséquence, nul n'est besoin de spécifier une adresse de communication en C, puisque le select code que l'on utilise précise lui-même l'implantation de la carte. Inversement, si vous souhaitez placer la carte à une adresse différente, HP propose dans sa documentation les select codes correspondants.

La librairie n'offre pas moins de 32 routines qui permettent une gestion efficace du bus IEEE. Contrairement à NI, il suffit de passer l'adresse de l'appareil que l'on souhaite piloter, à la fonction appelée. Chez HP, l'adresse se calcule par :

(ISC × 100) + adresse primaire effective. Ainsi, notre Fluke positionné en 5 sera représenté par le chiffre 705.

Les fonctions de base, écriture et lecture, prennent les formes respectives suivantes pour expédier ou lire des chaînes de caractères (S pour string) :
IOOUTPUTS (705, "TRIGGER 2", 9) ;
char buffer [25] ; (Stockage des données)
int count = 20 ; /*Nombre de caractères à lire*/
IOENTERS (705, buffer, &count) ; /*Lecture des données*/
Certaines des fonctions HP permettent de formater directement les éléments reçus.
IOENTER, par exemple, convertit immédiatement la chaîne ASCII "– 4,565E1" en un nombre décimal de valeur – 45,65. Cette particularité offre l'avantage de supprimer des lignes de conversion dans le programme, mais présente l'inconvénient de ne pas pouvoir utiliser le DMA puisque le processeur intervient lors de la conversion ASCII-réel. On retrouve le même type de traitement avec la fonction IOENTERA qui traite des tableaux.
Chaque fonction retourne une erreur sous forme d'entier dont la valeur diffère d'une fonction à l'autre. Par exemple, IOEOI, qui valide ou dévalide le mode End Or Identify ne renvoie que deux valeurs possibles : NOERR ou ESEL qui signifie respectivement pas d'erreur et code de sélection invalide. Par contre, IOSTATUS qui teste l'état actuel de l'interface IEEE, ne retourne que NOERR, ESEL et ERANGE qui indique un dépassement de gamme autorisée.
Autrement dit, lorsque vous attendez un Service Request par la fonction IOSTATUS (ISC, SRQLINE, &status), le test de timeout n'existe pas. Si votre équipement tombe subitement en panne ou est débranché par erreur, la boucle s'exécute à l'infini ! Heureusement, quelques lignes de C permettent de contourner cet inconvénient. La gestion des erreurs utilisera la routine show–err, mise en œuvre d'une manière quasi-similaire à celle utilisée dans les programmes d'exemple pour la PCIIA.

Le programme HP.C

Plutôt que de détailler une à une les fonctions HP qui nous intéressent, penchons-nous sur le programme de la **figure 10**. L'en-tête du fichier débute par les inclusions des classiques headers. Pour les appels de la librairie CLHPIB.LIB (ou sa version renommée...), on ajoutera les

```
/*
  Exemple de programmation avec la carte HP 82335A
  HP.C, Christophe BASSO 1992.
*/

#include <dos.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include "cfunc.h"                          /* Déclaration des constantes HP-IB */
#include "chpib.h"                           /* Déclaration des fonctions HP-IB */

#define ISC            7L                 /* Select Code de l'interface 82335A */
#define FLUKE_ADD 705L                         /* Adresse complète du Fluke 45 */
#define SRQLINE        1

void fail(char *s) {
                        puts(s);
                        puts("Program aborted");
                        exit(EXIT_FAILURE);
                    }

void show_err(int error, char *message)
{
    if (error != NOERR)                                       /* Erreur GPIB ? */
    {
      printf("Error when executing <%s>\n\a",message);
      switch (error)
      {
        case EUNKNOWN : puts("Unknown error");break;
        case ESEL     : puts("Invalid select code or device address");break;
        case ERANGE   : puts("Value out of range");break;
        case ETIME    : puts("Timeout");break;
        case ECTRL    : puts("HP-IB must be controller");break;
        case EPASS    : puts("Pass control not permitted");break;
        case ENUMB    : puts("Invalid Number");break;
        case EADDR    : puts("Improper addressing");break;
      }
      puts("Program aborted");
      exit(EXIT_FAILURE);
    }
}

void cleanup(void)
{
/* ---------- Initialise le Fluke à son état de départ ------------------ */

   show_err(IOOUTPUTS(FLUKE_ADD,"*RST",4),"IOOUTPUTS");

/* ---------- Repasse le Fluke en mode local ---------------------------- */

   show_err(IOLOCAL(FLUKE_ADD),"IOLOCAL");

   puts("Program terminated");
}

void init_bus(void)
{
/* ---------- Initialise l'interface 82335A ----------------------------- */

   show_err(IORESET(ISC),"IORESET");

/* ---------- Valide le time out à 5 secondes --------------------------- */

   show_err(IOTIMEOUT(ISC,5.0),"IOTIMEOUT");

/* ---------- Passe tous les dispositifs dans un état connu ------------- */

   show_err(IOCLEAR(ISC),"IOCLEAR");

/* ---------- Verrouille les accès par panneau frontal ------------------ */

   show_err(IOLLOCKOUT(ISC),"IOLLOCKOUT");
}

void program_fluke(void)
{
   char *buf1;

/* ---------- Envoie la séquence d'initialisation du Fluke ---------------*/

   buf1 = "*RST; VDC; AUTO; TRIGGER 2; *SRE 16";
   show_err(IOOUTPUTS(FLUKE_ADD,buf1,strlen(buf1)),"IOOUTPUTS");
}

float read_fluke(void)
{
   int  status=0,                               /* Mot d'état du Fluke 45 */
        count=20,                          /* Nombre de caractères lus */
        i=0;
   char buffer[25];                        /* Stockage de ces caractères */

/* ---------- Déclenche la conversion du volt-mètre --------------------- */

   show_err(IOTRIGGER(FLUKE_ADD),"IOTRIGGER");

/* ---------- Demande la sortie de la valeur convertie ------------------ */

   show_err(IOOUTPUTS(FLUKE_ADD,"VAL1?",5),"IOOUTPUTS");

/* ---------- Teste l'arrivée du service request ------------------------ */

   while (!status) {
                     show_err(IOSTATUS(ISC,SRQLINE,&status),"IOSTATUS");
                     if (status == 0)
                     {
                       i++;
                       delay(10);
                       if (i == 500) fail("SRQ not asserted\a");
                     }
                   }

/* ---------- Récupère le mot d'état du Fluke 45 ------------------------ */

   show_err(IOSPOLL(FLUKE_ADD,&status),"IOSPOLL");

/* ---------- Vérifie la présence éventuelle d'une erreur --------------- */

   if (status != 0x50) fail("Fluke 45 error");

/* ---------- Lit la chaine de caractères en sortie du Fluke ------------ */

   show_err(IOENTERS(FLUKE_ADD,buffer,&count),"IOENTER");

/* ---------- Retourne la valeur convertie en float --------------------- */
```

Figure 10

```
    return((float)atof(buffer));
}

void main(void)
{
    int loop;

/* ---------- Initialise l'interface et le bus IEEE --------------------- */

    init_bus();

/* ---------- Programme le Fluke 45 ------------------------------------- */

    program_fluke();

/* ---------- Boucle sur dix mesures et affiche le résultat ------------- */

    for (loop=0;loop < 10;loop++) printf("Fluke 45 = % f\n\r",read_fluke());

/* ---------- Termine proprement le programme --------------------------- */

    cleanup();
}
```

Figure 10 suite

fichiers cfunc.h et chpib.h. Les # define permettent de définir une fois pour toutes les différentes adresses utilisées dans le source. La lettre L spécifie le type long, nécessaire au correct passage des paramètres.

Comme dans les précédents sources étudiés, la première des instructions consiste à initialiser le bus et l'interface IEEE, ce dont la routine init–bus se charge :
IORESET met le contrôleur IEEE dans un état similaire à celui qui suit une mise sous tension, avec notamment le timeout fixé à zéro. On force ce dernier à 5 secondes, par la fonction suivante, IOTIMEOUT.IOCLEAR passe tous les périphériques reliés au bus dans un état connu, tel que le ferait leur bouton marche-arrêt. Enfin, IOLLOCKOUT expédie la commande LLO qui obligera chaque appareil **une fois adressé** à dévalider les touches de son panneau frontal.

La séquence de programmation du Fluke tient en une simple ligne, exécutée par la routine program–fluke. Le contenu de la chaîne reste identique à celui des programmes précédents.

On peut désormais démarrer une boucle de dix lectures périodiquement affichées à l'écran au retour de la fonction read–fluke. Rien de bien nouveau dans cette routine qui déclenche le Fluke avec IOTRIGGER, puis attend l'arrivée du SRQ avec IOSTATUS pour lire le multimètre.

Nous l'avons souligné plus haut, IOSTATUS ne retourne pas de timeout, ce qui poser problème si l'appareil ne répond plus. Pour juguler cet inconvénient, la mise en œuvre d'une simple temporisation interrompt le programme si le Service Request n'arrive pas avant 5 secondes. On modifiera ces valeurs en fonction des besoins de chacun. Le SRQ effectivement validé, nous lançons le polling série par IOSPOLL qui nous fournit le mot d'état du Fluke. Si aucune erreur ne se manifeste, on procède à la lecture du multimètre puis la valeur convertie extérieurement, on revient au programme principal.
La boucle achevée, la fonction cleanup repasse le Fluke en mode local après l'avoir initialisé.

Autres fonctionnalités de l'interface HP

La carte ne supporte pas les interruptions en language C, alors qu'en Basic elle autorise des branchements avec la fonction IOPEN, comme le fait également la PCIIA. En fait, le Basic scrute automatiquement la présence d'une SRQ à la fin de chaque ligne de programme et branche éventuellement sur la sous-routine de gestion. Il ne s'agit donc pas d'une véritable interruption.
Un dernier mot sur IOSEND, qui permet d'expédier des commandes bas niveau sur le bus. On transmet ici des caractères, comme le fait également NI avec son IBCMD.

LA PC-488 DE CAPITAL-EQUIPMENT-CORPORATION

Distribuée par Keithley France, la carte C.E.C. nous a agréablement surpris comme en témoignent les lignes qui suivent...
De taille légèrement supérieure à la PCIIA, cette interface s'enfiche sur un connecteur huit bits et exploite les possibilités du circuit NEC, le µPD7210. Les deux disquettes livrées avec la carte contiennent tous les éléments pour développer ses programmes en BASIC, QuickBASIC, Turbo Pascal, C, Fortran et enfin, assembleur. Les nombreux fichiers d'exemples contribueront à simplifier la prise en main de la carte et de son driver associé.
L'accès au contrôleur via le PC s'effectue selon deux méthodes : en BASIC, les CALL passent directement par l'EPROM de l'interface, qui contient tout le logiciel nécessaire à son fonctionnement. En language C, il suffit de lier avec la librairie IEEE488.LIB, dont les prototypes de fonctions figurent dans le header IEEE-C.H. La PC-488 autorise la mise en œuvre de sous-routines fonctionnant en interruption, lorsqu'un Service Request apparaît. Nous proposons un tel programme, décrit un peu plus bas.



Programmation de la PC-488

Nous venons de le souligner, la carte CEC comme sa prédécesseur HP, ne réclame pas de driver résident pour fonctionner. Il suffit donc en C, d'ouvrir un fichier projet (.PRJ) qui contiendra le ou les programmes développés ainsi que la librairie IEEE488.LIB.
Si vous souhaitez modifier en cas de conflit l'adresse I/O de la CEC-488 (0x2B8 par défaut), il faudra rajouter un appel à la routine setport. Par exemple, setport (0,0x2A8), indique que la carte 0 se trouve désormais à l'adresse hexadécimale, 0x2A8.
Pour envoyer une chaîne de caractères à un instrument, il suffit de taper les lignes suivantes :
define FLUKE5 /*Adresse du multimètre Fluke*/
int status ; /*Etat du bus après l'envoi de l'ordre*/
send (FLUKE, "VDC ; AUTO", & status) ;
if (status ! = 0) error–ieee ("Send", status) ;
Vous remarquerez immédiatement l'absence du paramètre spécifiant le nombre de caractère à émettre. De plus, la fonction ne retourne rien, mais modifie lors de son déroulement la variable status qui à son retour, contiendra le code d'une éventuelle erreur. Si aucun problème n'est venu perturber l'appel de la fonction, status vaudra zéro. Au contraire de National-Instruments qui décrit en détail les différents états du bus et qui autorise donc une réaction précise, CEC ne met en œuvre que deux états de status, 0 ou 8, avec la dernière valeur qui indique un timeout. Seules d'autres fonctions, dont transmit, proposent quelques codes de plus.
Attention, tous les codes d'erreur ne sont malheureusement pas communs. La valeur 2 retournée par la fonction transmit ne signifie pas la même chose qu'un 2 renvoyé par receive.
La lecture d'une donnée ne présente pas plus de difficultés que les lignes précédentes :

```
# define FLUKE 5 /*Adresse du multimètre Fluke*/
char buffer [25] ; /*Stockage des données lues*/
int length, /*Nombre de caractères effectivement lus*/
status ; /*Etat du bus après l'envoi de l'ordre*/
enter (FLUKE, 20, & length, FLUKE, & status) ;
if (status ! = 0) error–ieee ("Send", status) ;
```

Le paramètre 20 spécifie le nombre de caractères à lire, et length contient au retour de la fonction le nombre de caractères effectivement lus.

Le programme CEC.C

Comme les exemples précédents, il permet d'afficher dix valeurs de tension à l'écran, avant de s'arrêter (**figure 11**).

La classique initialisation du bus débute le source. On notera l'absence de code retourné par la fonction initialize qui ne permet donc pas de déceler un éventuel défaut du contrôleur. Arrive l'initialisation complète du Fluke 45, qui vous est désormais familière. La boucle entame la succession de dix lectures de tension. Chaque demande de conversion commence par un déclenchement du multimètre. Ici, pas de fonction de haut niveau, il faut impérativement passer par la commande transmit ainsi que les divers ordres IEEE qui lui sont associés. Heureusement, le manuel contient toutes les indications pour mener à bien ces opérations. On constatera avec bonheur que transmit accepte directement les mnémoniques du bus (UNL, GET...) et évite au programmeur d'éplucher régulièrement la table de conversion IEEE/ASCII.

A présent, le Fluke doit déclencher une demande de service. La fonction srq() se charge d'avertir le programme de son arrivée en retournant un booléen : SRQ présent, la fonction est vraie. Ici également, comme HP, pas de gestion d'un timeout ou d'une quelconque erreur... En conséquence, une petite routine s'impose pour quitter le programme au cas où la ligne SRQ ne se trouve pas validée avant cinq secondes.

Le polling série ne produisant pas d'erreur, on peut alors lire la valeur délivrée par le Fluke et l'afficher à l'écran. Mesure suivante... Le programme s'achève par la réinitialisation du multimètre, suivie de son retour en local par la fonction transmit.

```
 /*
   Exemple d'utilisation des fonctions CEC,
   avec la carte CEC-488, fichier CEC.C
   Christophe BASSO 1992
*/

#include <stdlib.h>
#include <stdio.h>
#include <dos.h>

#include "ieee-c.h"      /* Déclaration des prototypes de fonctions CEC-488 */

#define BOARD0 0                /* Adresse primaire de l'interface CEC-488 */
#define FLUKE  5                /* Adresse primaire du multimètre Fluke 45 */

void fail (char *s)
{
   puts (s);                                /* Affiche le message d'erreur */
   puts ("Program aborted");
   exit (EXIT_FAILURE);
}

void error_ieee (char *message,int status)
{
   printf ("Error when executing <%s>\n\a",message);
   switch (status)
   {
    case 1  : puts ("Illegal command syntax"); break;
    case 2  : puts ("Tried to send data when not a talker"); break;
    case 4  : puts ("A quoted string or END was found in a LISTEN");
              puts ("or TALK list");
              break;
    case 8  : puts ("Timeout or device not responding"); break;
    case 16 : puts ("Unknown command"); break;
   }
   puts ("Program aborted");
   exit (EXIT_FAILURE);
}

void main (void)
{
   int  status,                            /* Résultat des opérations GPIB */
        loop,                                   /* Comptage des lectures */
        length,                              /* Nombre de caractères recus */
        i=0;                      /* Compteur de sortie en cas d'absence d'IRQ */
   char status_byte,                          /* Mot d'état du multimètre */
        buffer[25];                               /* Stockage des mesures */

/* ---------- Initialisation de la carte CEC-488 ------------------------- */

   initialize (BOARD0,0);

/* ---------- Envoi de l'initialisation du Fluke 45 --------------------- */

   send (FLUKE,"*RST; VDC; AUTO; TRIGGER 2; *SRE 16",&status);
   if (status != 0) error_ieee ("Send",status);

   for (loop=0;loop < 10;loop++)
   {

/* ---------- Déclenche le Fluke 45 et demande la mesure ---------------- */

     transmit ("UNL LISTEN 5 GET",&status);
     if (status != 0) error_ieee ("Transmit",status);

     send (FLUKE,"VAL1?",&status);
     if (status != 0) error_ieee ("Send",status);

/* ---------- Attend l'arrivée du Service Request ----------------------- */

     while (!(status = srq()))
     {
       if (status == 0)
       {
         i++;
         delay (10);
         if (i == 500) fail ("SRQ not asserted\a");
       }
     }
     i=0;

/* ---------- Déclenche un polling série -------------------------------- */

     spoll (FLUKE,&status_byte,&status);
     if (status != 0) error_ieee ("Spoll",status);
     if (status_byte != 0x50) fail ("Fluke 45 error");

/* ---------- Lit la mesure disponible ---------------------------------- */

     enter (buffer,20,&length,FLUKE,&status);
     if (status != 0) error_ieee ("Enter",status);

/* ---------- Convertit puis affiche la valeur -------------------------- */

     printf ("Fluke 45 = % f\n\r",(float)atof(buffer));
   }

/* ---------- Réinitialise le Fluke et le passe en local ---------------- */

   send (FLUKE,"*RST",&status);
   if (status != 0) error_ieee ("Send",status);

   transmit ("UNL LISTEN 5 GTL",&status);
   if (status != 0) error_ieee ("Transmit",status);

   puts ("Program terminated");
}
```

Figure 11

Le travail sous interruption

Nous vous l'annoncions un peu plus haut, la carte supporte en standard le déclenchement d'une interruption en présence d'un Service Request. Le programme que nous vous proposons apparaît en **figure 12**. Avant d'entamer sa description, un rapide rappel du fonctionnement des interruptions s'impose.

Les interruptions dans le PC

Lorsqu'un programme se déroule, il peut, en présence d'un évènement extérieur, interrompre momentanément la tâche en cours pour exécuter une sous-routine réagissant à cet évènement. La sous-routine terminée, le programme revient là où il se trouvait et reprend son exécution. On dit alors que le pro-

gramme accepte de travailler sous interruption.

Il existe différents types d'interruptions qui peuvent être logicielles (appel au BIOS...) ou bien matérielles (un périphérique demande l'attention du processeur ou présente un défaut...). Nous nous intéressons uniquement aux dernières qui exploitent les lignes du bus IRQ 2 à IRQ 7. Cependant, une ligne peut se partager entre plusieurs périphériques et les conflits sont fréquents. On vérifiera alors l'absence de toute ambiguité lors du choix d'une IRQ. Par exemple, les COM1 et 2 pilotent respectivement les IRQ 4 et 3. LPT 1 et 2 se réservent les IRQ 7 et 5...

Lorsque l'IRQ se manifeste, le processeur ne quitte la tâche en cours que si les interruptions ont été préalablement autorisées par le positionnement d'un bit dans le registre du µP (sauf certaines que l'on ne peut empêcher, les NMI). Il existe également des niveaux de priorité au sein de ces interruptions. Ainsi, lors du traitement d'une IRQ, l'arrivée soudaine d'une autre IRQ de priorité supérieure force le processeur à stopper la tâche en cours, pour s'occuper de cette seconde interruption. Il revient ensuite à la première, puis à la fin de cette sous-routine signale au processeur que son traitement s'achève pour finalement retourner au programme original. Si il s'agit d'interruptions de niveaux inférieurs, le contrôleur d'interruptions (en fait ils sont deux sur un AT) les empile et les exécute plus tard. Ce circuit se nomme le "Programmable Interrupt Controler", PIC en abrégé, fabriqué par INTEL sous la référence 8259A. Deux de ses registres nous concernent, le Command Register disponible à l'adresse 0x20 et le "Interrupt Mask Register", accessible en 0x21. Le premier sera utilisé par notre programme, uniquement pour spécifier la fin de notre sous-routine d'interruption par l'envoi du caractère EOI, 0x20 (End-Of-Interrupt). Le registre des masques permet par sa programmation d'autoriser ou d'interdire certaines interruptions par la mise respective à 1 ou 0 d'un bit dans ce registre. Dans notre exemple, en l'absence de port parallèle 2, nous programmerons la carte CEC-488 sur l'IRQ 3 et expédierons au PIC, en début de source, le mode de validation en conséquence. A la fin de notre logiciel, il faudra restaurer au PIC son mot de validation initial par une simple opération logique.

```
/*
  Exemple d'utilisation des fonctions CEC sous interruption,
  avec la carte CEC-488, fichier CEC_INTR.C
  Christophe BASSO 1992
*/

#include <stdlib.h>
#include <stdio.h>
#include <dos.h>

#include "ieee-c.h"      /* Déclaration des prototypes de fonctions CEC-488 */

void interrupt srq_handler(void);                    /* Routine d'interruption */
void interrupt (*old_vector)();      /* Sauvegarde du vecteur d'interruption */

#define BOARD0   0               /* Adresse primaire de l'interface CEC-488 */
#define FLUKE    5               /* Adresse primaire du multimètre Fluke 45 */

int  status,                                  /* Résultat des opérations GPIB */
     length,                                 /* Nombre de caractères recus */
     loop=0,                                      /* Comptage des lectures */
     mesure=0,                  /* Si 1, mesure disponible pour affichage */
     i=0,                     /* Compteur de sortie en cas d'absence d'IRQ */
     fluke_error=0;                      /* Indicateur d'erreur du Fluke 45 */
char status_byte,                                /* Mot d'état du multimètre */
     buffer[25];                                     /* Stockage des mesures */

void fail (char *s)
{
  puts (s);                                    /* Affiche le message d'erreur */
  puts ("Program aborted");
  exit (EXIT_FAILURE);
}

void error_ieee (char *message, int status)
{
  printf ("Error when executing <%s>\n\a",message);
  switch (status)
  {
    case 1  : puts ("Illegal command syntax"); break;
    case 2  : puts ("Tried to send data when not a talker"); break;
    case 4  : puts ("A quoted string or END was found in a LISTEN");
              puts ("or TALK list");
              break;
    case 8  : puts ("Timeout or device not responding"); break;
    case 16 : puts ("Unknown command"); break;
  }
  puts ("Program aborted");
  exit (EXIT_FAILURE);
}

void cleanup (void)
{

  char ch;

/* ---------- Restore le vecteur d'interruption initial ----------------- */

  setvect(0x0B,old_vector);

/* ---------- Restore le masque des interruptions original -------------- */

  ch = inportb (0x21);     /* Récupère le masque des interruptions actuel */
  ch |= ~ 0xF7;                                          /* Masque l'IRQ3 */

  outportb (0x21,ch);               /* Repasse la nouvelle valeur du masque */
}

void install (void)
{
  char ch;

/* ---------- Sauvegarde l'ancient vecteur d'interruption --------------- */

  old_vector = getvect (0x0B);

/* ---------- Installe la fonction appelée lors de l'interruption -------- */

  setvect (0x0B,srq_handler);

/* ---------- Configure la carte IEEE pour les interruptions en SRQ ------ */

  outportb (0x2BA,0x40);

/* ---------- Programme le registre du PIC pour autoriser l'IRQ 3 -------- */

  ch = inportb (0x21);     /* Récupère le masque des interruptions actuel */
  ch &= 0xF7;                                         /* Démasque l'IRQ 3 */
  outportb (0x21,ch);               /* Repasse la nouvelle valeur du masque */
}

void interrupt srq_handler(void)
{

/* ---------- Réautorise les interruptions matérielles ------------------ */

  enable ();

/* ---------- Déclenche un polling série -------------------------------- */

  spoll (FLUKE,&status_byte,&status);
  if (status_byte != 0x50) fluke_error = 1;

/* ---------- Lit la mesure disponible ---------------------------------- */

  enter (buffer,20,&length,FLUKE,&status);

/* ---------- Signale au programme principal la présence d'une mesure ---- */

  mesure = 1;

/* ---------- Incrémente la valeur du compteur de mesures --------------- */

  loop ++;

/* ---------- Réinitialise la temporisation de l'SRQ -------------------- */

  i = 0;

/* ---------- Signale au PIC la fin de l'interruption ------------------- */

  outportb (0x20,0x20);
}

void main (void)
{

/* ---------- Installe la routine de sortie ----------------------------- */
```

Figure 12

```
    atexit (cleanup);

/* ---------- Initialisation de la carte CEC-488 ------------------------ */

    initialize (BOARD0,0);

/* ---------- Installe le gestionnaire d'interruption ------------------- */

    install ();

/* ---------- Envoi de l'initialisation du Fluke 45 --------------------- */

    send (FLUKE,"*RST; VDC; AUTO; TRIGGER 2; *SRE 16",&status);
    if (status != 0) error_ieee ("Send",status);

    while (loop < 10)                              /* Boucle infinie */
    {

/* ---------- Déclenche le Fluke 45 et demande la mesure ---------------- */

      transmit ("UNL LISTEN 5 GET",&status);
      if (status != 0) error_ieee ("Transmit",status);

      send (FLUKE,"VAL1?",&status);
      if (status != 0) error_ieee ("Send",status);

/* ---------- Attend la disponibilité de la mesure ---------------------- */

      while (!mesure)
      {
        i++;
        delay (10);
        if (i==500) fail ("SRQ not asserted\a");
      }

/* ---------- Teste les éventuelles erreurs ----------------------------- */

      if (fluke_error) fail ("Fluke 45 error");
      if (status != 0) error_ieee ("Error in SRQ routine",status);

/* ---------- Affiche le résultat de l'acquisition ---------------------- */

      printf ("Fluke 45 = % f\n\r",(float)atof(buffer));

      mesure = 0;
    }

/* ---------- Réinitialise le Fluke et le passe en local ---------------- */

    send (FLUKE,"*RST",&status);
    if (status != 0) error_ieee ("Send",status);

    transmit ("UNL LISTEN 5 GTL",&status);
    if (status != 0) error_ieee ("Transmit",status);

    puts ("Program terminated");
}
```

Figure 12 suite

Lors de l'arrivée de l'interruption, le PIC retourne au µP un numéro d'identification (0x0B pour l'IRQ 3), qui permet à ce dernier de trouver le point d'entrée dans la table des vecteurs d'interruptions. Celle-ci, occupant les premiers octets de la RAM, fournit au µP l'adresse de la sous-routine à laquelle il doit se brancher en fonction de l'identifiant reçu :

IRQ	Identifiant	Adresse du vecteur (Contient l'adresse des sous-routines d'interruption)
2	10 (0x0A)	0000:0028
3	11 (0x0B)	0000:002C
4	12 (0x0C)	0000:0030
5	13 (0x0D)	0000:0034
6	14 (0x0E)	0000:0038
7	15 (0x0F)	0000:003C

Cet emplacement (0000 :002C dans notre cas), contient l'adresse de la routine à exécuter en présence de l'IRQ. Notre travail de programmation va donc consister à modifier temporairement le contenu de la case IRQ 3, bien entendu après la sauvegarde préalable de la table des vecteurs initiale. A la fin du programme, une fonction du C se chargera de la remettre en place.

CEC-INTR.C

Atexit() permet de terminer un programme par une sous-routine, installée par cette fonction (interruption 22 h du DOS). Il s'agit de cleanup(), qui restaure les divers registres modifiés en début de logiciel. Attention, lors de l'exécution du programme, une sortie par Ctrl-Break redonne immédiatement la main à DOS, sans lancer cleanup, soit en ne restaurant pas la table des vecteurs originale...

Le programme continue avec la fonction install(), à laquelle il incombe de sauver puis programmer, tous les registres et tables énoncés précédemment.

Il ordonne également à la CEC-488, de déclencher une interruption en présence d'une demande de service.

Les classiques lignes d'initialisation puis de déclenchement arrivent derrière. A ce moment, on boucle sur l'attente de l'interruption, qui lors de son déroulement, positionne à 1 la variable mesure, indiquant la disponibilité d'un résultat. Il va de soi qu'utiliser une boucle en attendant l'arrivée de l'interruption constitue une hérésie puisque le principe veut que le programme se déroule normalement, et intervienne à tout moment lorsqu'on le lui signale ! Nous avons cependant volontairement choisi cette méthode pour simplifier au maximum le source dont le but premier est d'illustrer la mise en œuvre d'interruptions sous IEEE, et non de réaliser des performances...

Lorsque l'IRQ3 se manifeste, la routine srq–handler() entre en jeu. Elle comporte le mot clé enable() qui autorise les autres interruptions matérielles et essaie ainsi de perturber le moins possible le système. Le polling série renseigne sur l'état du Fluke 45, et initialise éventuellement une variable en cas d'erreur. On procède ensuite à la lecture de la tension convertie, et finalement, après indication au PIC de la fin d'interruption, on retourne au programme principal.

Vous remarquerez que la sous-routine srq–handler ne comporte aucun appel au DOS. En effet, celui-ci n'est pas réentrant. Cela signifie qu'on ne peut lors du déroulement d'un appel au DOS (en l'occurence notre interruption), relancer une autre de ses fonctions . Par exemple, ne vous amusez pas à afficher à l'écran par printf la valeur de la tension convertie dans la sous-routine... Sinon, préparez la séquence des touches magiques... ou même de l'interrupteur marche-arrêt.

La programmation en style Hewlett-Packard

CEC propose également un "character I/O driver" du nom de CECHP.EXE que l'on lance sous la ligne DOS. Il suffit, en C, d'inclure la ligne suivante pour communiquer correctement avec l'interface via ce handler :

FILE*ieee ;

ieee = fopen ("ieee", "r") ; /* Ouvre le descripteur de fichier ieee*/

setbuf (ieee, NULL) ; /*Supprimer la fonction buffer associée au Fonctionnement du flux ieee*/

Les divers appels en style hp transitent ensuite selon le modèle ci-après :

fprintf (ieee, "OUTPUT 5 ; VDC ; TRIGGER 2\n") ;

CEC offre quelques 25 fonctions en style HP, dont SRQ ? qui autorise la détection d'un Service Request.



La programmation en HP BASIC

Comme nous l'avons souligné précédemment, les éditeurs de software offrent aux utilisateurs de nombreux compilateurs ou interprèteurs, leur permettant de développer dans le langage de leur choix. Lorsque sur un PC on souhaite travailler en véritable HP Basic, peu de solutions existent, si ce n'est l'achat d'une carte spéciale auprès de Hewlett-Packard. Cette interface, équipée d'un processeur de la famille 68 000 et d'un contrôleur IEEE, permet à l'utilisateur de travailler directement sous l'environnement d'un HP 9000 et de mettre au point ses programmes en HP Basic.

Pour les gens qui possèdent déjà une interface IEEE (l'une des trois étudiées dans l'article par exemple) et désirent cependant écrire en HP Basic, la solution commercialisée par le constructeur américain TransEra devrait les satisfaire.

Le HT Basic

Ce logiciel recrée sur un PC l'environnement de travail mis en œuvre sur un HP 9000. Il offre une compatibilité complète avec des logiciels déjà écrits en HP Basic sur ces machines. De plus, en chargeant des drivers appropriés, HT Basic peut piloter de nombreux contrôleurs IEEE présents sur le marché.

Après le chargement du driver correspondant à votre carte IEEE, voici quelques lignes de programme qui vous démontrent la facilité d'exploitation de ce type de langage :

```
10 OUTPUT 705 ; "*RST ; TRIGGER 2"
20 TRIGGER 705
30 OUTPUT 705 ; "VAL 1 ?"
40 ENTER 705 ; Data$
50 PRINT Data$
60 END
```

HT Basic offre également de nombreuses possibilités de tracés graphiques ainsi que de puissantes fonctions de traitement du signal et d'analyse numérique (transformées de Fourrier, filtrage digital, convolution...), fournies par la librairie mathématique TransEra. Un fichier d'aide accessible à tout instant lors du développement renseigne immédiatement le développeur sur la signification et la syntaxe d'une commande HT Basic.

La puissance et la simplicité de ce Basic s'expriment au travers du travail sous interruption qui se traduit par les lignes suivantes :

```
10 Hpib = 7
20 ON INTR Hpib GOSUB Srq-handler
30 Mask = 2
40 ENABLE INTR Hpib ; Mask !
```

Programme la carte pour une interruption en cas de SRQ.

Ainsi, lorsqu'un SRQ arrive, le programme se branche sur la sous-routine après avoir terminé l'exécution de la ligne en cours. Le nombre de lignes nécessaires pour réaliser cette fonction se passe de commentaires...

La seule chose que l'auteur puisse regretter concerne l'environnement de travail de HT Basic qui reprend avec toute sa pauvreté (même l'affichage monochrome !), celui que l'on rencontre sur HP 9000. Rien à voir avec l'EDI de Borland ! Cependant, les gens qui transitent d'une machine 9000 sur un PC équipé du HT Basic ne verront aucune différence et pourront immédiatement commencer à développer.

TransEra distribue une disquette de démonstration qui, en principe, vante les qualités de son produit. En fait, cette version d'évaluation est quasiment inexploitable pour quelqu'un qui souhaite véritablement tester le HT Basic ! En effet, d'une part les sources que vous écrirez ne pourront être sauvés, mais en plus, au bout de 30 minutes, le logiciel vous mettra à la porte. Nous pensons que le constructeur devrait s'inspirer des versions de démo genre PSpice, en fournissant un programme d'évaluation entièrement exploitable (sauvegarde, pas de limite en temps) mais dont les sources développés ne pourraient excéder un certain nombres de lignes.

Ainsi, l'éventuel client travaille avec un produit aux performances non tronquées, ce qui lui permet de l'apprécier à sa juste valeur et de se diriger, pourquoi pas, vers la version complète.

TransEra, distribué par ALTIS aux Ulis, fabrique également des contrôleurs IEEE que nous aurons prochainement l'occasion de décrire.

Les générateurs de code

Nous ne pouvions publier un article consacré à la programmation des contrôleurs IEEE sans aborder rapidement les logiciels générateur de code. Le principe de ces produits repose sur des panneaux de configuration qui comportent une face avant représentative des fonctions de l'instrument concerné (voir photographie). Le développeur, selon ses besoins, actionne alors les boutons de son choix pour valider ou dévalider les commutateurs de l'équipement concerné. Une fois sa sélection achevée, le logiciel produit une ligne de code directement exploitable par un compilateur. Ainsi, les bibliothèques d'instruments étant disponibles auprès des éditeurs de software, la programmation se trouve simplifiée à l'extrême, puisque la nécessité de connaître la syntaxe inhérente à chaque appareil disparaît.

National-Instruments commercialise LabWindows, qui permet le pilotage de nombreux équipements sous QuickBasic et Microsoft C. Ce logiciel intègre également un puissant interface graphique, qui autorise très simplement l'écriture de logiciels aux performances et présentations professionnelles (voir photos). La mise en place de boutons, sélecteurs ou encore écrans genre oscilloscope, enrichit la présentation et facilite la prise en main du produit final par des personnes techniquement incompétentes. Associé aux diverses librairies de calcul (Standard and Advanced Analysis Library), LabWindows effecute de nombreux traitements offerts par des DSP sur des signaux échantillonnés sous bus GPIB, VXI ou encore RS232 (FFT, FHT, convolution, filtres digitaux...).

La firme Capital Equipment Corporation propose son logiciel Co-Operator disponible sous forme de programme résident de type pop-up. Ainsi, en développant sous son éditeur favori, l'utilisateur fait apparaître à tout instant un panneau de fonctions décrivant l'instrument de son choix. Il peut ensuite rapidement sélectionner les interrupteurs désirés, pour immédiatement inclure dans son source la ligne de code générée.

De plus, Co-Operator possède la faculté d'expédier la commande sélectionnée à l'instrument piloté, ce qui permet d'apprécier le résultat des opérations, sans passer par toutes les étapes de compilation et linkage. Co-Operator supporte de nombreux compilateurs et interpréteurs, comme Turbo C, Turbo Pascal, ou encore GWbasic. CEC vend également d'autres produits, dont le pilotage de ses cartes IEEE-488 sous Windows 3 avec Visual Basic.

VIEWDAC de Keithley Asyst est un programme spécialement développé pour fonctionner sur des 386 et 486. Il produit des applications professionnelles, destinées à automatiser les

acquisitions de données. L'environnement graphique, associé à l'utilisation de la souris, en font un produit particulièrement attractif pour ceux qui développent des systèmes exploités ensuite par des opérateurs non techniciens. La puissance de VIEWDAC autorise le fonctionnement simultané de plusieurs séquences d'acquisition distinctes, supportant ainsi les possibilités multi-tâches de la famille des micro processeurs 386/486.

CONCLUSION

Les programmes proposés au cours de cet article devraient, nous l'espérons, vous aider à rapidement développer des applications de test, autour des contrôleurs IEEE de votre choix. Bien que les appels de fonctions diffèrent selon les constructeurs, la philosophie de mise en œuvre d'un programme d'acquisition reste la même et permet ainsi de passer facilement d'une carte à l'autre. Tous les sources décrits sont disponibles sur le serveur ERP et autorisent ainsi leur téléchargement immédiat.

Nous donnerons les références complètes des fabricants et distributeurs lors de notre prochain article consacré à la présentation des contrôleurs IEEE du commerce.

Christophe BASSO



Bibliographie

NI-488.2 MS-DOS Software Reference Manual, **National Instruments**

CEC 488, Programming and Reference Manual, **Capital Equipment Corporation**

Using the HP-IB Interface and Command Library, **Hewlett-Packard**

Developing a Lab Windows Instrument Driver, A.N.022, **National Instruments**

Programming with NI-488 SoftWare, A.N.002, **National Instruments**

Sur les interruptions

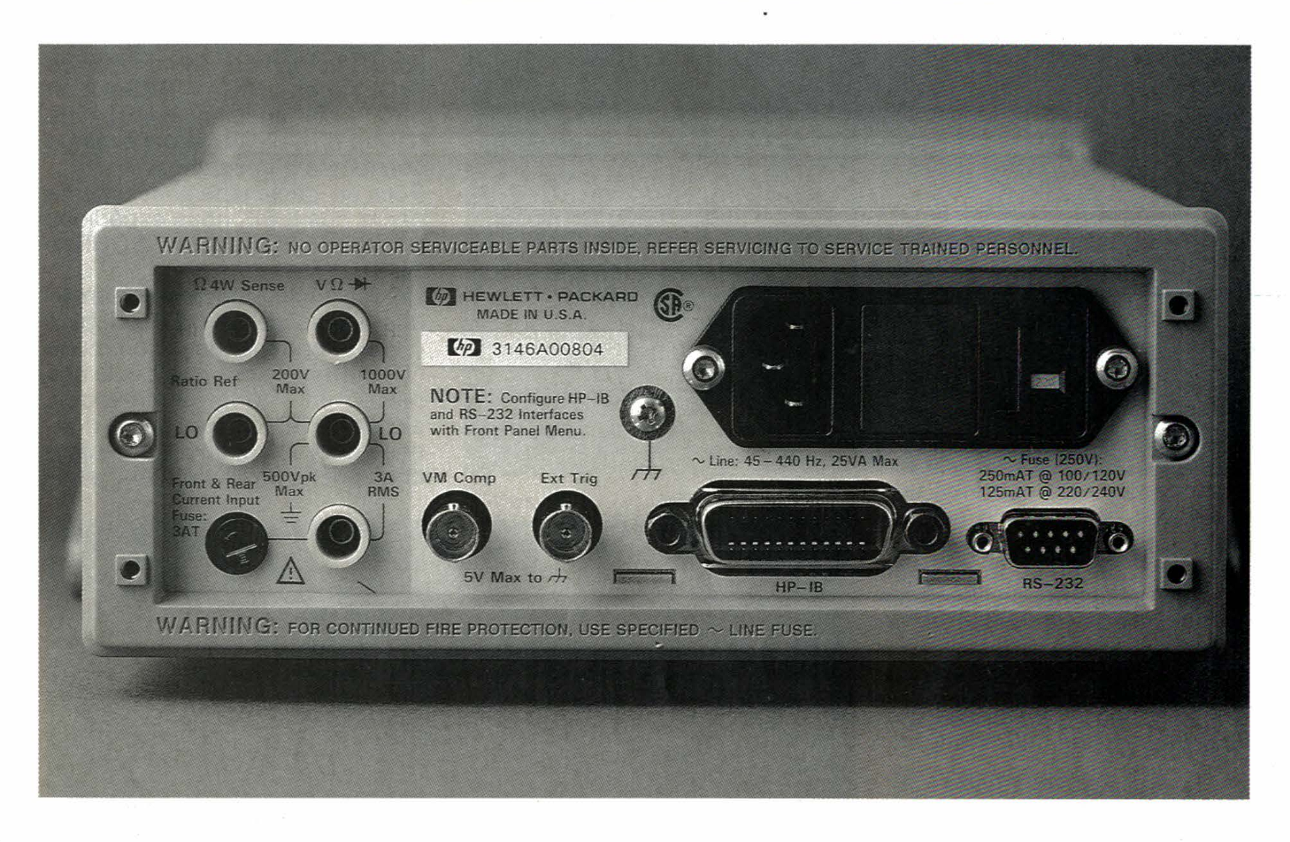
Serial Communication With Turbo C, **The C users Journal April 1991**

Comment fonctionnent les interruptions, **Micro-Systèmes nº 128**

La Bible PC, programmation système, **Editions Micro Application**

Le guide Peter Norton du Pogrammeur sur PC & PS/2, **Microsoft Press**

La programmation du HP34401A



Comme souligné dans la publication précédente, le HP34401A supporte le langage SCPI (prononcer SKIPI) sur les bus série et parallèle. L'un des avantages de cette caractéristique réside dans le peu de modification que nécessite le remplacement d'un instrument par un autre, également compatible avec ce mode de communication. De plus, une fois la syntaxe assimilée par l'utilisateur, le passage à un autre équipement ne posera aucune difficulté et réduira les coûts de développement en conséquence.

L'exemple proposé

Pour illustrer le pilotage IEEE-488.2 du 34401A, le logiciel que nous avons développé en C permet de tracer, en association avec un générateur HP8116A, la courbe de réponse d'un filtre passe-bas de type KROHN-HITE 3202. La manipulation consiste à mesurer, puis stocker point par point, l'atténuation provoquée par le filtre dont la fréquence de coupure sera arbitrairement fixée à 300 Hz. Le programme complet se scinde en trois sous-routines, destinées à gérer individuellement les deux instruments, ainsi que le contrôleur IEEE du PC. Il s'agit, pour ce dernier, de la carte HP82335A décrite ailleurs dans ce même numéro.

LE PROGRAMME PRINCIPAL FILTRE.C

Il apparaît en **figure 1**. L'organigramme retenu consiste à initialiser les divers périphériques (HP34401A, HP8116A et HP82335A) puis balayer la gamme de fréquence choisie point par point, avant de stocker les résultats sur disque dur. Dans notre cas, nous programmerons le générateur pour délivrer des signaux sinusoïdaux dont la fréquence évoluera de 10 à 1 000 Hz, par pas de 10 Hz. Ces paramètres dépendent des variables START, STOP et STEP, définies dans le header (entête) HPIB.H de la **figure 6 b**.

Après les appels aux sous-routines d'initialisation (init-bus(), init-8116() et init-34401()), la boucle de mesure proprement dite, débute. Chaque incrémentation en fréquence du générateur entraîne une lecture du 34401 qui retourne la valeur d'atténuation mesurée. En fait, la routine init-34401 comprend une instruction, qui envoyée à la fréquence de démarrage, stocke le premier échantillon comme 0 dB de référence. Les mesures ultérieures calculent alors des valeurs relatives.

L'instruction suivante range les résultats sur le disque dur, dans un fichier portant le nom et le chemin définis au début du source (≠ define path-file). On notera l'instruction de tabulation entre chaque chiffre (\t) qui rend le fichier texte compatible avec le format Excel. Une fois la boucle achevée, la fonction end-

```
/*
   Exemple de programmation du HP34401A avec la carte HP 82335A
   FILTRE.C, Christophe BASSO 1992.
*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include "hpib.h"                    /* Déclaration des fonctions du HP82635A */
#include "hp34401a.h"                /* Déclaration des fonctions du HP34401A */
#include "hp8116a.h"                  /* Déclaration des fonctions du HP8116A */

#define path_file "c:\\language\\tc\\34401a\\filt_acq.dat"

void main(void)
{
   int   frq;                               /* Valeur de la fréquence en cours */
   float val_db;                         /* Résultat de la mesure des décibels */
   FILE *dat_file;                            /* Pointeur du fichier de données */

/* ---------- Ouverture du fichier de données ------------------------------*/

   if ((dat_file = fopen(path_file,"wt")) == NULL )
        fail("Unable to open destination file\a");

   init_bus();                       /* Initialise l'interface et le bus IEEE */
   init_8116();                           /* Initialise le générateur HP8116A */
   init_34401();                        /* Initialise le multimètre HP34401A */

/* ---------- Lance l'acquisition des points de fréquence et tension ----- */

   for (frq=START;frq <= STOP;frq+=STEP)
   {

      inc_8116(frq);     /* Incrémente la fréquence de sortie du générateur */
      val_db = read_34401();                  /* Demande la mesure du multimètre */

/* ---------- Sauvegarde des données sur le disque et affichage ---------- */

      fprintf(dat_file,"%d\t% 2.3f\n",frq,val_db);
      printf("Atténuation de % 2.3f dB à %d Hz\n",val_db,frq);
   }

   end_34401();                             /* Signale la fin de l'acquisition */
   cleanup(dat_file);                       /* Termine proprement le programme */
}
```

Figure 1

34401 signale à l'utilisateur la fin de la mesure. Enfin, la fonction cleanup() se charge de libérer les instruments et de refermer le fichier.

La commande du multimètre, HP34401A.C

Nous ne détaillerons pas les routines de communication IEEE, puisque l'article sur la programmation des cartes PC l'aborde en détail dans ce même numéro. HP34401A.C se trouve en **figure 2 a**, alors que son fichier header apparaît en **figure 2 b**.
Le sous-programme init-34401, consiste à envoyer des ordres de configuration au multimètre, de telle sorte qu'il accepte les mesures de décibels et les déclenchements par le bus.
Les instructions *RST et *CLS font partie des commandes communes définies par la norme IEEE-488.2 (objet de notre prochaine publication...). La première se charge de la remise à zéro de l'instrument, tandis que la seconde efface tous ses registres d'état. Pour mesurer les décibels, HP indique la marche à suivre :
– Passer en mode voltmètre alternatif : **CONF**igure : **VOLT**age : AC
– Sélectionner la fonction mathématique dB : **CALC**ulate : **FUNC**tion DB
– Valider cette fonction décibel : **CALC**ulate : **STAT**e ON
Démarrée en début de programme, cette séquence initialise le 0 dB pour les basses fréquences. Le multimètre effectue alors les mesures qui suivent, de manière relative.
La norme SCPI demande d'expédier au minimum la partie majuscule du mnémonique, la seconde en minuscule étant facultative.
Comme nous souhaitons tracer la courbe de réponse du filtre assez rapidement, la commande "**VOLT**age : AC : **RES**olution MAX", demande au 34401A de passer en 4 digits, ce qui diminue son temps d'intégration.
Pour un déclenchement externe par le bus IEEE, nous expédions la ligne "**TRIG**ger : **SOUR**ce BUS".
Un déclenchement externe implique l'envoi d'un GET ou bien d'une commande commune *TRG. Dans le premier cas le bus passe en mode commandes, alors qu'il retourne en mode données pour le second. Le déclenchement d'un instrument sup-

```
/*
   Routines de gestion du multimètre HP34401A
   HP34401A.C, Christophe BASSO 1992
*/

#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <dos.h>

#include "cfunc.h"                          /* Déclaration des fonctions HP-IB */

#include "hp34401a.h"                  /* Déclaration des fonctions du HP34401A */
#include "hpib.h"                      /* Déclaration des fonctions du bus HPIB */

void init_34401(void)
{
   char *buf1;

/* ---------- Envoie la séquence d'initialisation du HP34401A------------- */

   show_err(IOOUTPUTS(HP34401A,"*RST",4),"IOOUTPUTS HP34401A");

/* ---------- Remet à zéro tous les mots d'état -------------------------- */

   show_err(IOOUTPUTS(HP34401A,"*CLS",4),"IOOUTPUTS HP34401A");

/* ---------- Configure le HP34401A en volt-mètre AC --------------------- */

   buf1 = "CONF:VOLT:AC";
   show_err(IOOUTPUTS(HP34401A,buf1,strlen(buf1)),"IOOUTPUTS HP34401A");

/* ---------- Passe en résolution de 4 digits ---------------------------- */

   buf1 = "VOLT:AC:RES MAX";
   show_err(IOOUTPUTS(HP34401A,buf1,strlen(buf1)),"IOOUTPUTS HP34401A");

/* ---------- Sélectionne la fonction mathématique ----------------------- */

   buf1 = "CALC:FUNC DB";
   show_err(IOOUTPUTS(HP34401A,buf1,strlen(buf1)),"IOOUTPUTS HP34401A");

/* ---------- Autorise la fonction mathématique choisie ------------------ */

   buf1 = "CALC:STAT ON";
   show_err(IOOUTPUTS(HP34401A,buf1,strlen(buf1)),"IOOUTPUTS HP34401A");

/* ---------- Programme le déclenchement par le bus IEEE ----------------- */

   buf1 = "TRIG:SOUR BUS";
   show_err(IOOUTPUTS(HP34401A,buf1,strlen(buf1)),"IOOUTPUTS HP34401A");

/* ---------- Passe en retard de déclenchement automatique --------------- */

   buf1 = "TRIG:DEL:AUTO ON";
   show_err(IOOUTPUTS(HP34401A,buf1,strlen(buf1)),"IOOUTPUTS HP34401A");
}

float read_34401(void)
{
   int  status=0,                                  /* Mot d'état du HP34401A */
        count=20,                            /* Nombre de caractères lus */
        i=0;
   char buffer[25];                              /* Stockage de ces caractères */

/* ---------- Remet à zéro tous les mots d'état -------------------------- */

   show_err(IOOUTPUTS(HP34401A,"*CLS",4),"IOOUTPUTS HP34401A");

/* ---------- Valide le bit "operation complete" (bit 0, event register) - */

   show_err(IOOUTPUTS(HP34401A,"*ESE 1",6),"IOOUTPUTS HP34401A");

/* ---------- Déclenche le SRQ lorsque le bit 5 passe à 1 ---------------- */

   show_err(IOOUTPUTS(HP34401A,"*SRE 32",7),"IOOUTPUTS HP34401A");
```

Figure 2 a

```
/* ---------- Passe en mode "Wait for Trigger" ---------------------------- */

   show_err(IOOUTPUTS(HP34401A,"INIT",4),"IOOUTPUTS HP34401A");

/* ---------- Déclenche la conversion du multimètre ---------------------- */

   show_err(IOTRIGGER(HP34401A),"IOTRIGGER HP34401A");

/* ---------- Assure la synchronisation du multimètre -------------------- */

   show_err(IOOUTPUTS(HP34401A,"*OPC",4),"IOOUTPUTS HP34401A");

/* ---------- Teste l'arrivée du service request ------------------------- */

   while (!status) {
                     show_err(IOSTATUS(ISC,SRQLINE,&status),"IOSTATUS");
                     if (status == 0)
                     {
                        i++;
                        delay(10);
                        if (i == 500) fail("SRQ not asserted\a");
                     }
                 }

/* ---------- Récupère le mot d'état du HP34401A ------------------------- */

   show_err(IOSPOLL(HP34401A,&status),"IOSPOLL");

/* ---------- Vérifie la présence éventuelle d'une erreur ---------------- */

   if (status != 0x60) fail("HP34401A error");

/* ---------- Demande la sortie de la valeur convertie ------------------- */

   show_err(IOOUTPUTS(HP34401A,"FETC?",5),"IOOUTPUTS HP34401A");

/* ---------- Lit la chaine de caractères en sortie du HP34401A ---------- */

   show_err(IOENTERS(HP34401A,buffer,&count),"IOENTER");

/* ---------- Retourne la valeur convertie en float ---------------------- */

   return((float)atof(buffer));
}

void end_34401(void)
{
   char *buf1;

/* ---------- Emet un beep pour signaler la fin d'acquisition ------------ */

   buf1 = "SYST:BEEP";
   show_err(IOOUTPUTS(HP34401A,buf1,strlen(buf1)),"IOOUTPUTS HP34401A");

/* ---------- Affiche le mot "Termine" ----------------------------------- */

   buf1 = "DISP:TEXT \"Termine\"";
   show_err(IOOUTPUTS(HP34401A,buf1,strlen(buf1)),"IOOUTPUTS HP34401A");

   sleep(2);
}
```

Figure 2 a suite

```
/* Déclaration des protoypes de fonctions du HP34401A */

#ifndef __HP34401
#define __HP34401

#define HP34401A  706L          /* Adresse complète du multimètre HP34401A */
#define SRQLINE      1

void  init_34401(void);                        /* Initialise le HP34401A */
float read_34401(void);          /* Retourne la valeur lue par le multimètre */
void  end_34401(void);                  /* Signale la fin d'acquisition */

#endif
```

Figure 2 b

portant le SCPI suit le diagramme dessiné en **figure 3**. Cette chronologie particulière autorise des déclenchements manuels ou automatiques, des lectures multiples pour chaque déclenchement (jusqu'à 50 000) et la possibilité d'inclure un retard avant chaque mesure. Nous laissons le choix de cette dernière option au multimètre, avec la ligne "**TRIG**ger : **DEL**ay : AUTO ON".

Nous arrivons ensuite à la lecture du multimètre, développée par la routine read-34401.

Avant de débuter toute opération, la commande *CLS remet à zéro tous les mots d'état de l'instrument. Nous travaillerons en Service Request afin de signaler au contrôleur IEEE la disponibilité d'une mesure. Nous utiliserons à cet effet le bit 0 du registre événement standard (Standard Event Register), conformément à l'architecture de la **figure 4**. Cette nouvelle définition des mots d'état découle de la norme 488.2, définie par l'IEEE en 1987.

Nous ne nous appesantirons pas plus sur cette particularité puisque nous publierons un article consacré à ce standard le mois prochain. Pour déclencher un SRQ avec le bit "operation complete", nous utilisons l'instruction *SRE 32 qui valide la demande de service lorsque le bit 5 du mot d'état (status byte) passe à 1. Nous l'évoquions précédemment (voir **figure 3**), il faut passer le multimètre en mode "wait for trigger" avant tout déclenchement. Il s'agit de l'instruction INIT. Le déclenchement par le bus exploite la fonction IOTRIGGER, spécifique à l'interface HP82335A.

L'instruction *OPC demande à l'instrument de basculer à 1 le bit "operation complete" à la fin de sa mesure. A ce moment, le Service Request doit se manifester après un délai dépendant du temps d'intégration sélectionné.

La boucle while (! status) attend que l'instrument demande l'attention du contrôleur dans une fenêtre inférieure à un maximum de 5 secondes. Passée cette limite, une erreur stoppe l'exécution du programme en signalant à l'utilisateur l'absence de SRQ.

Si l'opération précédente s'est déroulée correctement, le polling du HP34401A ne doit déceler aucune faille et autoriser la lecture de la mesure. La commande FETC ?, transfère le résultat

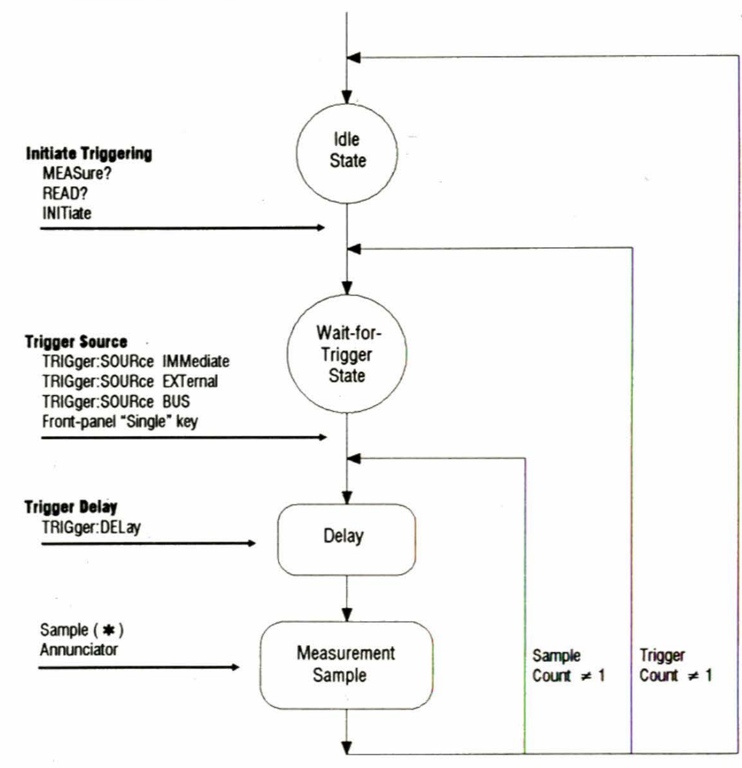


Figure 3

stocké en mémoire vers le tampon de sortie IEEE, alors sollicité par la routine HPIB IOENTERS. La chaîne de caractères convertie en chiffres décimaux retourne finalement vers le programme principal, pour être sauvegardée puis affichée à l'écran.
La dernière sous-routine end-34401 interpelle l'utilisateur par un bip sonore et lui témoigne de la fin des opérations par le message "Termine".
Le fichier d'en-tête contient l'adresse du multimètre HP ainsi que les déclarations des fonctions utilisées ailleurs dans le programme.

Le contrôle du générateur HP8116A.C

Quelques lignes suffisent pour programmer ce générateur de fonctions selon la configuration souhaitée (**figures 5 a** et **5 b**). La routine inc-8116 (int) transforme l'entier expédié par le programme principal témoin de la fréquence en cours (variable locale frq), en une chaîne de caractères assimilables par le générateur. Si value vaut 100, la chaîne envoyée sera alors : FRQ 100 Hz.
Le HP8116A accepte l'un des deux procédés signalant la fin de message : <CR> <LF> ou EOI. Nous utiliserons le premier, d'où la demande de dévalidation du second dans le programme HPIB.C par l'instruction : IOEOI (0). Attention, si votre carte expédie les deux ordres, le HP8116A ne réagira pas à vos injonctions.

La gestion de l'interface, HPIB.C

Rien de très original dans ce source, proposé en **figures 6 a** et **6 b**, qui se charge de gérer les éventuelles erreurs apparues sur le bus (show-err) et d'initialiser la carte (init-bus). Appelée en fin d'acquisition, la routine cleanup() se charge de libérer tous les instruments et de refermer le fichier de données, ouvert dans le programme principal.

Le résultat des mesures

Le fichier traité par Excel se métamorphose en une courbe, telle celle de la **figure 7**. Le graphique s'aplatit en fin de mesure, témoin du plancher de bruit, inhérent au filtre employé.
Selon le dispositif à tester, la modification des variables START, STOP et STEP devrait vous permettre aisément d'adapter le logiciel à votre cas.

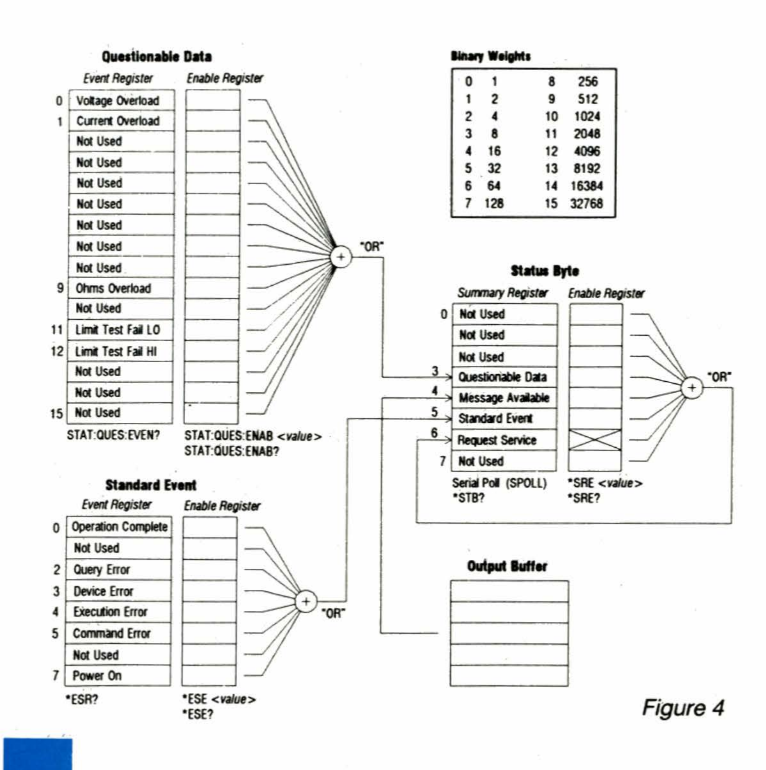


Figure 4

```
/*
   Routines de gestion du générateur HP8116A
   HP8116A.C, Christophe BASSO 1992
*/

#include <stdio.h>
#include <string.h>

#include "cfunc.h"                      /* Déclaration des fonctions HP-IB */

#include "hp8116a.h"               /* Déclaration des fonctions du HP8116A */
#include "hpib.h"                 /* Déclaration des fonctions du bus HPIB */

void init_8116(void)
{
   char *buf1;

/* ---------- Envoie la séquence d'initialisation du HP8116A ------------- */

   buf1 = "M1, CTO, W1, TO, CO, DO, HIL 2.5V, LOL -2.5V";
   show_err(IOOUTPUTS(HP8116A,buf1,strlen(buf1)),"IOOUTPUTS HP8116A");
   sprintf(buf1,"FRQ %d Hz",START);
   show_err(IOOUTPUTS(HP8116A,buf1,strlen(buf1)),"IOOUTPUTS HP8116A");

/*
   M1            : mode Normal
   CTO           : pas de modulation
   W1            : sortie Sinus
   CO            : sortie normale non-inversée
   DO            : sortie validée
   TO            : pente de déclenchement off
   FRQ START Hz  : fréquence de START Hertz
   AMP 5V        : amplitude 5 volts peak-peak
*/
}

void inc_8116(int value)
{
   char buf1[12];

   sprintf(buf1,"FRQ %d Hz",value);          /* Elabore la chaîne à expédier */
   show_err(IOOUTPUTS(HP8116A,buf1,strlen(buf1)),"IOOUTPUTS HP8116A");
}
```

Figure 5 a

```
/* Déclaration des protoypes de fonctions du HP8116A */

#ifndef __HP8116
#define __HP8116

#define HP8116A 709L             /* Adresse complète du multimètre HP34401A */
#define STEP     10                   /* Définit l'incrément de fréquence */
#define START    10                     /* Définit le départ du balayage */
#define STOP   1000                       /* Définit la fin du balayage */

void init_8116(void);               /* Initialisation du générateur HP8116A */
void inc_8116(int);                     /* Incrémentation de la fréquence */

#endif
```

Figure 5 b

Remarques importantes de programmation

Si comme nous, vous utilisez la carte HP82335A, Hewlett-Packard livre les drivers qui contiennent les fonctions de pilotage (*.LIB) nécessaires à la commande de l'interface en langage C. On trouve notamment deux fichiers .H, CHPIB.H et CFUNC.H qui déclarent les prototypes développés par HP. Au cas où vous compileriez le programme proposé ci-dessus, il faudrait procéder à une légère manipulation. En effet, CHPIB.H contient un bout de source en C qui gère les erreurs survenues sur le bus. Or si l'on inclut plusieurs fois de suite ce fichier header (dans HP34401A.C, HPIB.C et HP8116A.C), le linker produit une erreur puisque la fonction précitée se trouve dupliquée ! Il suffit alors de reporter uniquement les déclarations de constantes livrées par CHPIB.H dans CFUNC.H pour lever le tracas. On inclut finalement CFUNC.H au lieu des deux headers comme proposé par le constructeur.

Voir également dans notre article sur la programmation des cartes PC la mise en garde pour ceux qui développent sous Turbo C avec un fichier projet.

La ligne de programmation **CAL**-**C**ulate : **FUNC**tion DB permet de mesurer des décibels relatifs et de travailler à l'aide d'une référence à 0 dB. Cependant, si ailleurs dans votre programme vous changez de fonction entre deux mesures de décibels, la valeur d'origine du 0 disparaît, rendant inexploitables les résultats affichés ! Nous avons rencontré le problème car nous souhaitions au départ mesurer la fréquence à chaque point produit par le générateur... Si vous désirez vous lancer dans cette configuration, il faut utiliser la chronologie suivante dans la routine init-34401 :

static float zero-db ;
char buffer [50] ;
CONFigure : **VOLT**age : AC
CALCulate : **FUNC**tion DB
CALCulate : **STAT**e ON
CALCulate : DB : **REF**erence 0
INIT
FETC ?

Récupérer par le bus la valeur représentative du 0 dB à la fréquence voulue, par exemple 23.7169 dB, puis la stocker dans la variable zéro-db.

Finalement, la boucle consistera à effectuer :

CONFigure : **FREQ**uency
/* Passe en fréquencemètre */

```
/*
   Gestion du bus HPIB et de la carte HP82335A
   HPIB.C, Christophe BASSO 1992
*/

#include <stdio.h>
#include <stdlib.h>

#include "cfunc.h"                              /* Déclaration des fonctions HP-IB */

#include "hpib.h"
#include "hp34401a.h"
#include "hp8116a.h"

void fail(char *s) {
                        puts(s);
                        puts("Program aborted");
                        exit(EXIT_FAILURE);
                   }

void show_err(int error, char *message)
{
   if (error != NOERR)                                          /* Erreur GPIB ? */
   {
      printf("Error when executing <%s>\n\a",message);
      switch (error)
      {
         case EUNKNOWN : puts("Unknown error");break;
         case ESEL     : puts("Invalid select code or device address");break;
         case ERANGE   : puts("Value out of range");break;
         case ETIME    : puts("Timeout");break;
         case ECTRL    : puts("HP-IB must be controller");break;
         case EPASS    : puts("Pass control not permitted");break;
         case ENUMB    : puts("Invalid Number");break;
         case EADDR    : puts("Improper addressing");break;
      }
      puts("Program aborted");
      exit(EXIT_FAILURE);
   }
}

void cleanup(FILE *fich)
{
/* ---------- Initialise le multimètre HP à son état de départ ----------- */

   show_err(IOOUTPUTS(HP34401A,"*RST",4),"IOOUTPUTS");

/* ---------- Repasse le multimètre et le générateur en mode local ------- */

   show_err(IOLOCAL(HP34401A),"IOLOCAL HP34401A");
   show_err(IOLOCAL(HP8116A),"IOLOCAL HP8116A");

/* ---------- Referme le fichier de données ------------------------------ */

   if (fclose(fich) == EOF) fail("Unable to close destination file\a");

   puts("Program terminated");
}
void init_bus(void)
{

/* ---------- Initialise l'interface 82335A ------------------------------ */

   show_err(IORESET(ISC),"IORESET");

/* ---------- Valide le time out à 5 secondes ---------------------------- */

   show_err(IOTIMEOUT(ISC,5.0),"IOTIMEOUT");

/* ---------- Dévalide la ligne EOI sur l'interface IEEE ----------------- */

   show_err(IOEOI(ISC,0),"IOEOI");

/* ---------- Passe tous les dispositifs dans un état connu -------------- */

   show_err(IOCLEAR(ISC),"IOCLEAR");

/* ---------- Verrouille les accès par panneau frontal ------------------- */

   show_err(IOLLOCKOUT(ISC),"IOLLOCKOUT");
}
```

Figure 6 a

```
/* Déclaration des prototypes de gestion du bus HPIB */

#ifndef __HPIB
#define __HPIB

#define ISC 7L                          /* Select Code de l'interface 82335A */

void show_err(int,char *);                      /* Traitement des erreurs HPIB */
void cleanup(FILE *);                   /* Repasse tous les appareils en local */
void init_bus(void);                                 /* Initialise le bus HPIB */
void fail(char *);                                    /* Sortie en cas d'erreur */

#endif
```

Figure 6 b

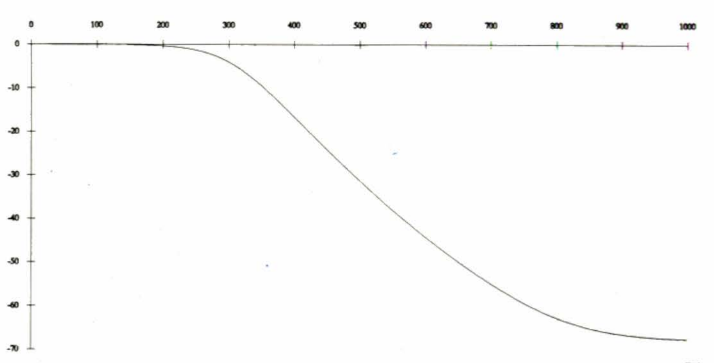


Figure 7

INIT /*Déclenche une mesure */
FETC ? /*Résultat de la mesure de fréquence*/
... /* Exploitation du résultat */
CONFigure : **VOLT**age : AC
/* Passe en voltmètre AC */
CALCulate : **STAT**e ON
/* Les indicateurs Math et dB s'allument */
/* .... Forme la chaîne de caractères en incluant la variable zéro-db .... */
sprintf (buffer, "**CALC**ulate : DB : **REF**erence %f", zéro-db) ;
IOOUTPUTS (HP34401A, buffert, strlen (buffer)) ; /* Expédie la chaîne au HP34401A */
INIT
/* Déclenche une mesure */
FETC ?
/* Récupère la valeur référencée au zéro dB précédent */

Si vous mettez cette option en œuvre, n'oubliez pas alors d'inclure les lignes de programmation du déclenchement (TRIG:SOUR et TRIG:DEL), en début de routine read-34401. En effet, le fait de changer de fonction sur le multimètre, dévalide l'option retenue avec la précédente et nécessite une mise à jour à chaque lecture...

En cas de problèmes détectés par le HP34401A (signal sonore acccompagné du témoin error), la commande **SYST**em:***ERR**or? vous retourne, en clair, le numéro d'erreur suivi du message explicatif (tous deux normalisés SCPI).

CONCLUSION

La programmation du HP34401A en langage SCPI ne pose pas de problème majeur. Cependant, pour éviter une perte de temps aux débutants, nous conseillons d'acquérir le HP82335U qui offre un environnement de développement HPIB sous Windows 3 absolument complet. Des fenêtres d'émission ou de réception permettent de dialoguer successivement avec plusieurs appareils IEEE et de leur expédier tous les messages possibles (en mode commandes ou données). L'analyse des erreurs devient un jeu d'enfant et l'écriture des programmes un vrai régal. En fait, malgré quelques perles de sueur au départ, le pilotage en SCPI du 34401A nous a apporté beaucoup de satisfaction.

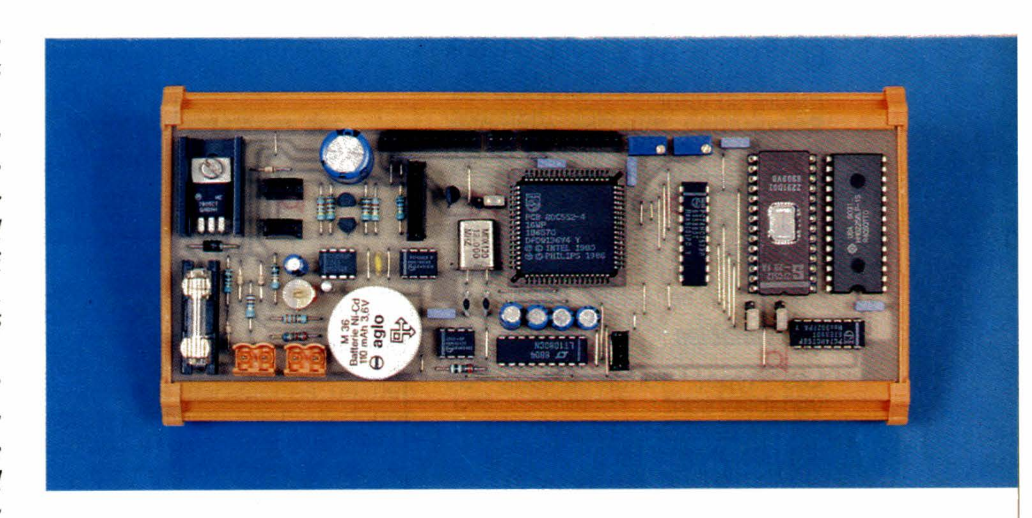
Christophe BASSO

Bibliographie

Le bus IEEE-488, **Electronique Radio-Plans** n° 534.
La programmattion des cartes IEEE pour PC, **Electronique Radio-Plans** n° 536.

Un module logiciel I2C en langage "C"

Dans le numéro de février notre confrère P. Morin vous a présenté les grandes lignes du langage C. Plus modestement nous allons revenir aux notions simples de ce mode de programmation pour vous présenter le même logiciel (DRIVER I2C), que celui présenté en mars écrit en Assembleur, mais transposé maintenant en "C". Pourquoi le langage "C" ? Le langage C fait parti des langages évolués comme le Pascal, l'ADA... En effet, il existe des langages orientés machine tel que l'assembleur et des langages hors machine tel que le C. Le premier a le défaut d'être lié à un type de machine et de n'être pas très plaisant. Le second peut s'adapter sur toutes les machines à partir du moment où on lui associe un compilateur.



De plus en plus, le langage C est utilisé dans les différentes sociétés pour n'importe quel projet (programmation d'une carte, logiciels...). C'est pourquoi, nous avons décidé de nous mettre à la page en utilisant le langage C créé par Kernighon et Richie dans les années 80. Seulement, ce langage a déjà ses successeurs : le C+ et le C++. Attendons le C+++ ou le langage D. Mais, seulement, connaissez-vous ce langage ? Si oui, attendez le prochain article. Si non, nous allons vous donner un récapitulatif afin de le comprendre.
Surtout, ne nous en veuillez pas d'expliquer des choses simples afin que tout le monde puisse facilement comprendre.
Les érudits nous pardonneront de ne pas avoir cherché la complexité : ce n'est qu'une question de pédagogie...

PRESENTATION GENERALE

Un programme C est un ensemble de fichiers et de fonctions. Chaque fichier peut être considéré comme autonome mais non indépendant des autres.
Chaque fonction permet de décomposer un projet en petits programmes, d'utiliser des résultats intermédiaires, de dissimuler des détails qui n'ont pas besoin d'être connus de la structure générale du programme.
Chaque fichier (ou fonction) est identique à une boîte noire qui a pour mission d'effectuer des opérations et qui est utilisable par d'autres fichiers sans se préoccuper de ce qui est fait à l'intérieur.
Le fichier général s'appelle MAIN ( ). C'est le pied !
La **figure 1** nous donne un aperçu d'une structure générale d'un programme en C.
La partie 1, placée en tête (HEADER = ENTETE = BLABLA.H) permet d'inclure tous les fichiers déjà existants dans ce langage (bibliothèques au sens logiciel...).
La partie 2 définie toutes les variables nécessaires.
La partie 3 est le programme principal dans lequel un appel à différentes fonctions est effectué. Ce programme principal est délimité entre deux accolades.
Les fonctions constituent la partie 4.
Comme vous pouvez le constater, chaque fonction peut avoir ses propres HEADERs. Il est de bon ton de tous les disposer en tête afin de s'y retrouver. De même, contrairement à une certaine logique qui pourrait faire

croire qu'on présente MAIN ( ) d'abord puis les fonctions ensuite, il est de coutume de présenter d'abord toutes les fonctions, considérées comme des boîtes noires (sous-programmes) puis le programme principal qui les appelera sachant que vous les connaissez (c'est ce que nous nous évertuerons à faire lors de la présentation de nos routines). Le langage C n'a donc rien inventé dans la structure d'un programme. Alors, à vos crayons afin de devenir les programmeurs du futur.

Remarque : tous les exemples se réfèrent au logiciel présenté en fin d'article.

Dans tout programme écrit en langage évolué, il est nécessaire de déclarer les variables et les constantes.

Les variables

Les types fondamentaux de variables sont :
- caractère CHAR 8 bits
- entier INT 32 bits
- réel simple précision FLOAT 32 bits
- réel double précision DOUBLE 64 bits.

Par exemple :
INT X, Y ; où X et Y sont les variables.
Le type entier peut être précisé court (SHORT), long (LONG) et positif (UNSIGNED).
Toutes ces variables doivent être déclarées avant leur utilisation. Elles peuvent être initialisées si elles sont dynamiques, statiques ou externes. Les variables dynamiques non initialisées prennent des valeurs parasites ou sont mises à zéro par défaut. Par exemple :
UNSIGNED CHAR slave, arrêt ; où slave et arrêt sont les variables.

Les constantes

Les constantes fournissent des noms symboliques à des valeurs qui ne varient pas. La valeur est substituée au nom à chaque fois qu'il est rencontré dans le programme.
Par exemple :
define ON 1 où la constante ON vaudra toujours 1.

Les opérateurs

Les différents opérateurs sont :
- "-" moins unitaire ou soustraction
- + addition
- * multiplication
- / division
- % modulo (reste de la division entière)

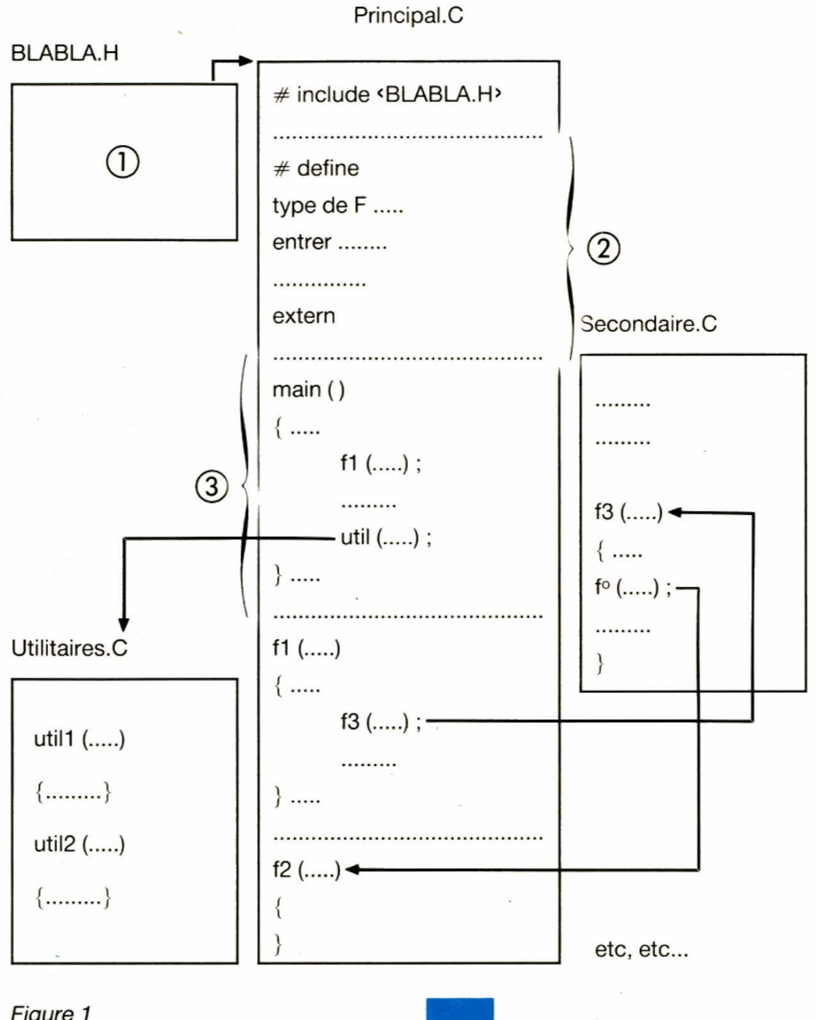


Figure 1

Les différentes relations sont :
- > supérieur
- < inférieur
- >= supérieur ou égal
- <= inférieur ou égal
- == égal
- != différent
- ! négation
- && et logique
- !! ou logique
- & et
- ! ou
- ˆ ou exclusif
- complémentation unitaire
- << décalage à gauche
- >> décalage à droite

L'opérateur d'incrémentation est ++, celui de décrémentation est −−.

Par exemple :
1) CAR == 0x21 qui se lit :
Si la variable "CAR" vaut à cet instant la valeur 21 en hexa, alors une action est à entreprendre.
2) fin_tr != CORRECT qui se lit :
Si la variable "fin de transmission" baptisée "fin_tr" est différente de CORRECT alors nous effectuerons un traitement spécifique.
3) S1DAT = pt_md++ signifie que S1DAT = (pt_md) + 1 ce qui veut dire en bon français que S1DAT vaut la valeur de la variable "pt_md" incrémentée de un.
De la même façon qu'il existait un jeu d'instructions en assembleur et des instructions d'assemblage, il existe des instructions C et de compilation C.

Les instructions C

Les instructions sont de la forme :
expression ;
De ce fait, une expression devient une instruction si elle est suivie d'un ";"

L'instruction conditionnelle

Comme dans beaucoup de langages, ce type d'instruction est connue sous la forme :
SI condition alors instruction 1 sinon instruction 2.
En langage C, ceci s'exprime de la façon :
IF (expression) instruction 1 ;
ELSE instruction 2 ;
Par exemple :



IF (car == 0x21)
ligne_rr = ligne_rr + 10 ;
boite_menu () ;
Ceci signifie que si la variable "CAR" vaut 21 en hexa, alors la valeur baptisée ligne_rr augmentera de 10 et on fera appel à la fonction boite_menu.

L'instruction à choix multiples
Elle permet d'effectuer un choix direct parmi plusieurs propositions : en langage C, elle s'exprime :
SWITCH (expression)
case <constante1> : <instruction1>
case <constante2> : <instruction2>
default : <instructiond>
Une remarque importante est à apporter. Si expression donne pour résultat la constante1, instruction1 est exécutée ainsi que les instructions suivantes. Pour cela, il faut mettre un break à la fin de l'instruction pour chaque cas pour les rendre exclusifs. Par exemple :
SWITCH (car)
case 0x38 : modif_param_eau_propre () ; break ;
case 0x3F : no_cur () ; break ;
default : no_cur () ; break ;
Ceci signifie que si la variable "CAR" vaut 38 en hexa, on fera appel à la fonction qui modifie le paramètre eau propre baptisé "modif_param_eau_propre". Par contre, si la variable CAR vaut 3F en hexa, alors on fera appel à la fonction no_cur. Mais, si la variable CAR ne vaut aucune de ces deux valeurs, on fera appel dans tous les cas à la fonction no_cur.

Boucles itératives et conditionnelles

Structure itérative TANT QUE.
L'expression est évaluée, exécution de l'instruction et évaluation de l'expression jusqu'à ce que sa valeur devienne fausse.
WHILE (expression) <instruction>
Par exemple :
WHILE (fin_tr != CORRECT)
if (fin_tr == NO_ANSWER_SLAVE)
fin_tr = RAZ ;
STA = ON ;
Ceci signifie que TANT QUE la variable fin de transmission n'est pas CORRECTE, il faut exécuter le jeu d'instructions suivant.

Structure itérative FAIRE... TANT QUE
L'instruction est exécutée puis l'expression est évaluée. Si le test est positif, l'instruction est réexécutée, jusqu'à ce que l'expression devienne fausse.
DO <instruction>
WHILE (expression) ;

Structure itérative FOR
FOR (expression1 ; expression2 ; expression3) instruction
L'expression1 correspond à une initialisation et est exécutée une fois. L'expression2 détermine la continuation ou la terminaison de la boucle. L'expression3 agit sur les conditions.

Fonctions et structures des programmes

Ces fonctions permettent de décomposer un projet en petits programmes, d'utiliser des résultats intermédiaires, de dissimuler des détails qui n'ont pas besoin d'être connus.
Le but d'une fonction est de lui donner des paramètres ou rien en entrée afin qu'elle nous donne quelque chose en sortie. Pour cela, il est nécessaire de lui affecter des paramètres qui sont appelés arguments.
La différence entre un appel et la définition de la fonction s'effectue par la présence ou non du ";".
F (arg1, arg2) définie la fonction.
F (argv, argy) ; est l'instruction appelant F
Dans cet exemple, deux paramètres sont données en entrée. Lorsque l'on appelle la fonction F, on lui donne les deux paramètres que l'on désire (ici arg1 et arg2). Automatiquement, à l'appel de la fonction, arg1 remplacera argv et arg2 remplacera argy dans la fonction.
Ainsi, lors d'un autre appel, il est possible de passer comme paramètre tata ou toto où tata prendra la valeur de argv et toto la valeur de argy.
A chaque appel, on obtiendra des valeurs différentes suivant les paramètres d'entrée.

Argument d'une fonction

Tous les arguments sont transmis par valeur et non par adresse. La fonction appelée ne peut donc pas modifier les variables du programme. Par contre, pour les tableaux, il serait encombrant de passer tous les paramètres et c'est donc l'adresse du premier élément qui est passée.
Voici un exemple :
La fonction boite_sous_menu (unsigned char colon) doit être appelée avec un argument de type caractère. L'appel se fera de la façon suivante :
boite_sous_menu (8) ;
Les variables à l'intérieur des fonctions sont internes et non visibles de l'extérieur.

Allocation mémoire des variables

Une variable est automatique si on désire y accéder uniquement dans le bloc où on l'a déclarée.
AUTO int x ;
Une variable est externe si elle est déclarée à l'extérieur de toute fonction. Elle ne pourra être accédée que dans la source où elle a été déclarée.
EXTERN unsigned char backup ;
Les variables statiques sont permanentes (elles peuvent être internes ou externes). Elle sont seulement utilisables à l'intérieur du fichier source dans lequel elles ont été déclarées.
STATIC int x ;
Les variables registre sont employées pour les variables très utilisées car elle ne font pas d'accès mémoire puisqu'elles utilisent les registres du microprocesseur.
REGISTER int x ;

Pointeurs et tableaux

Les tableaux

Un tableau est une structure de données de dimension finie dont les éléments sont consécutifs et de même type. Un tableau se déclare de la façon suivante en C :
Par exemple :
unsigned char mtd 20 ; où on définit 20 entrées pour le master transceiver data.
L'opérateur signifie "tableau de".
Sa taille peut être obtenue avec un sizeof () qui fournit sa dimension en octets. De plus, les éléments du tableau sont mtd 0, mtd 1...

Les pointeurs

Un pointeur est une variable qui contient l'adresse d'un objet. Il se déclare de la façon suivante.
Par exemple :
unsigned char *pt_mtd ; où on définit un pointeur sur le master transceiver data.
Pour accéder à l'aide d'un pointeur, il est nécessaire de faire :
pt_mtd = &x ;
Ceci signifie qu'on afffecte l'adresse de x à pt_mtd. De ce fait, pt_mtd pointe sur x car cette variable contient l'adresse de x.

```
/*****************************************************************/
/*                                                               */
/*                       I 2 C 1 . H                             */
/*                     -------------                             */
/*  DATE : 14/04/91                    UPDATE   : 14/04/91       */
/*                                                               */
/*****************************************************************/

/*
Fonction    : ini_i2c1().
Traitement : initialisation de la liaison serie I2C.
Entree      :
Sortie      :

Fonction    : gere_i2c1()
Traitement : gestion d'une communication I2C.
Entree      : code de gestion de la liaison I2C ( 0 pour bouclage sur liaison
              et 1 pour sortie meme en cas d'erreur )
Sortie      : status du bus :     NO_ANSWER        0x20
                                  UNCORRECT_TRANS  0x30
                                  BUS_LOST         0x38
                                  BUS_ERROR        0x00
                                  CORRECT          0xF0
*/

extern    ini_i2c1() ;
extern    unsigned char gere_i2c1(unsigned char code) ;


/***************************************/
/*    I N I _ I 2 C                    */
/*                                     */
/*  initialise la liaison I2C          */
/***************************************/

ini_i2c()
{
    SDA  =  ON ;              /* broches SDA et SCL au niveau haut */
    SCL  =  ON ;
    S1CON = ENS1_NOTSTA_NOTSTO_NOTSI_AA_CR0 ;
    ES1 =  ON ;                  /* IT I2C autorisee    */
}

/*--------------------------------------------------------------*/

/***************************************/
/*    G E R E _ I 2 C                  */
/*                                     */
/* Gere la communication sur le bus i2c*/
/*                                     */
/*  Prend le bus I2C, et attend que la */
/*  communication soit terminee, puis  */
/*  renvoie le code correspondant      */
/***************************************/


 unsigned char gere_i2c(unsigned char code)
 {
    RT2 = OFF ;
    fin_tr = RAZ ;
    STA = ON ;
         while ( fin_tr != CORRECT )
           {
              if ( fin_tr == NO_ANSWER_SLAVE )
                 {
                    fin_tr = RAZ ;
                    STA = ON ;
                 }
           }
    RT2 = ON ;
    return(fin_tr) ;
 }


/*------------------------------------------------------------------*
/*                                                                  *
/*                     M O D U L E   I 2 C                          *
/*                                                                  *
/* DATE : 22/07/91                            UPDATE : 22/07/91     *
/*                                                                  *
/* VERSION : 2.0                                                    *
/*                                                                  *
/*  Ce module contient les fonctions de gestion de la liaison i2c   *
/*                                                                  *
/*------------------------------------------------------------------*


/* Pour une version 8XC552 12WP avec I2C a 100Kb les valeurs ci dessous sont*

#define          ENS1_NOTSTA_STO_NOTSI_AA_CR0            0xD5
#define          ENS1_NOTSTA_NOTSTO_NOTSI_AA_CR0         0xC5
#define          ENS1_NOTSTA_NOTSTO_NOTSI_NOTAA_CR0      0xC1
#define          ENS1_STA_NOTSTO_NOTSI_AA_CR0            0xE5

/* Pour une version 8XC552 -4 avec I2C a 63Kb les valeurs ci dessous sont

#define          ENS1_NOTSTA_STO_NOTSI_AA_CR0            0x57
#define          ENS1_NOTSTA_NOTSTO_NOTSI_AA_CR0         0x47
#define          ENS1_NOTSTA_NOTSTO_NOTSI_NOTAA_CR0      0x43
#define          ENS1_STA_NOTSTO_NOTSI_AA_CR0            0x67
*/

#define  ON  1
#define  OFF 0
#define  TRUE  1
#define  NO_ANSWER_SLAVE 0x20
#define  UNCORRECT_TRANSMISSION 0x30
#define  BUS_LOST 0x38
#define  BUS_ERROR 0x00
#define  CORRECT 0xF0
#define  RAZ 0xFF

data unsigned char hadd ;
unsigned char    slave, numbyte, backup, fin_tr, status, arret ;
unsigned char    mtd[20],mrd[10] ;

unsigned char    *pt_mtd, *pt_mrd;
/*--------------------------------------------------------------*/
```

```
/***************************************/
/*    F C T _ 0 0                      */
/*                                     */
/*  Erreur sur le bus I2C              */
/***************************************/

fct_00()
{

      S1CON = ENS1_NOTSTA_STO_NOTSI_AA_CR0 ;      /* Raz SI */
                                                  /* Mise a 1 STO,AA */
      fin_tr = BUS_ERROR ;
}

/*--------------------------------------------------------------*/

/*********************************************/
/*    F C T _ 0 8                            */
/*                                           */
/*  Condition de depart effectuee, l'adresse */
/*  du peripherique est envoyee              */

/*********************************************/

fct_08()
{

      S1DAT = slave ;                     /* Chargement SLA+R/W ds tampon */
      S1CON = ENS1_NOTSTA_NOTSTO_NOTSI_AA_CR0 ;
                                          /* Raz SI */
      pt_mtd = mtd ;              /* pointeurs sur buffer initialises */
      pt_mrd = mrd ;

      backup = numbyte ;          /* Sauvegarde valeur initiale */
}

/*--------------------------------------------------------------*/

/***************************************/
/*    F C T _ 1 8                      */
/*                                     */
/*  acquittement recu, envoi du 1er    */
/*  caractere                          */
/***************************************/

fct_18()

{
     S1DAT = *pt_mtd++; /* pt_mtd pointe sur le buffer de transmission */
                        /* on envoie le 1er caractere, puis on pointe  */
                        /* sur le suivant                              */

     S1CON = ENS1_NOTSTA_NOTSTO_NOTSI_AA_CR0 ;
                                          /* Raz SI et AA */
}


/*--------------------------------------------------------------*/

/****************************************/
/*    F C T _ 2 0                       */
/*                                      */
/*  adresse esclave en ecriture envoyee */
/*  pas d'acquittement recu. On envoie  */
/*  une condition de stop et fin_tr     */
/*  indique que la communication s'est  */
/*  mal terminee                        */
/****************************************/

fct_20()
{
        S1CON= ENS1_NOTSTA_STO_NOTSI_AA_CR0 ;   /* Mise a 1 STO, raz SI*/
        numbyte=backup ;         /* numbyte est reinitialise a sa valeur */
                                 /* de depart  */
        fin_tr = NO_ANSWER_SLAVE ;

}

/*--------------------------------------------------------------*/

/****************************************/
/*    F C T _ 2 8                       */
/*                                      */
/*  acquittement recu en reponse au 1er */
/*  octet transmis                      */
/****************************************/

fct_28()
{

    if (--numbyte == 0 )
        {
          /* plus d'octet a transmettre. On envoie une condition de stop */
         S1CON= ENS1_NOTSTA_STO_NOTSI_AA_CR0 ;
                                        /* Raz SI et mise a 1 de AA */
         fin_tr = CORRECT ;  /* fin de communication correcte  */
         arret = ON ;
        }
    else
        {
          /* un autre octet a transmettre */
         S1DAT = *pt_mtd++ ;
         S1CON= ENS1_NOTSTA_NOTSTO_NOTSI_AA_CR0 ;  /* Raz de SI  */
        }
}
```

Exemple de logiciel de gestion I2C (toutes les fonctions ne sont pas présentes).

Pointeurs et tableaux
Comme vous pouvez l'imaginer, on peut associer un pointeur afin d'accéder à certains éléments d'un tableau.
pt_mtd = md ; signifie qu'on positionne le pointeur pt_mtd sur le premier élément du tableau md. De cette façon, il devient possible d'effectuer des opérations sur les pointeurs.
Par exemple :
S1_DAT = *pt_mtd++ ;
Ceci se décompose en : S1_DAT = *pt_mtd puis pt_mtd++.
Comme pt_mtd pointe sur le premier élément du tableau, cela signifie que S1_DAT prend la valeur du premier élément du tableau md. Ensuite, on incrémente le pointeur qui va alors pointer sur le deuxième élément du tableau.
Ainsi, si nous avions placé cette instruction dans une boucle, nous aurions pu envoyé sur le bus I2C toutes les données à la suite comme le montre l'exemple de programme donné en annexe.

Blandine DELABRE

Et remerciements à D. Paret et J.-P. Billard pour leurs participations amicales.

```
/*-----------------------------------------------------------------*/

/****************************************/
/*    F C T _ 3 0                       */
/*                                      */
/*  pas d'acquittement recu en reponse  */
/*  au 1er octet de donnees transmis    */
/*  On arrete donc la communication     */
/****************************************/

fct_30()
{
   S1CON= ENS1_NOTSTA_STO_NOTSI_AA_CR0 ; /* envoi d'une condition de stop */
   fin_tr = UNCORRECT_TRANSMISSION ;     /* communication mal terminee */
}

/*-----------------------------------------------------------------*/

/****************************************/
/*    F C T _ 3 8                       */
/*                                      */
/*  Perte du bus lors d'un arbitrage    */
/****************************************/

fct_38()
{
   S1CON= ENS1_STA_NOTSTO_NOTSI_AA_CR0 ;  /* Raz de SI  */
   numbyte=backup ;      /* numbyte reprend sa valeur initiale */
   fin_tr = BUS_LOST ;
}

/*-----------------------------------------------------------------*/

/****************************************/
/*    F C T _ 4 0                       */
/*                                      */
/*  acquittement recu en reponse a      */
/*  l'envoi d'une adresse en lecture    */
/****************************************/

fct_40()
{
   S1CON= ENS1_NOTSTA_NOTSTO_NOTSI_AA_CR0 ;    /* Raz de SI  */

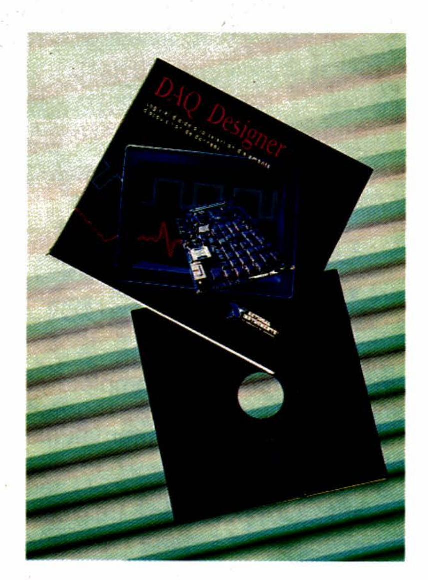
}

/*-----------------------------------------------------------------*/
etc.
```



DAQ Designer : 1er outil d'aide à la sélection d'éléments d'acquisition de données

National Instruments annonce un logiciel d'un type totalement nouveau : DAQ Designer. Gratuit et en français, DAQ Designer s'adresse à tous les développeurs qui souhaitent mettre en place un système d'acquisition de données sur ordinateur (PC/XT/AT/EISA et Micro Channel). Il intervient dans la phase de sélection des composantes du système d'acquisitioin, en éliminant la tâche fastidieuse de recherche dans les catalogues, et en assurant au développeur qui l'aura sélectionné une combinaison matériels/logiciels adaptée à son application de mesure.

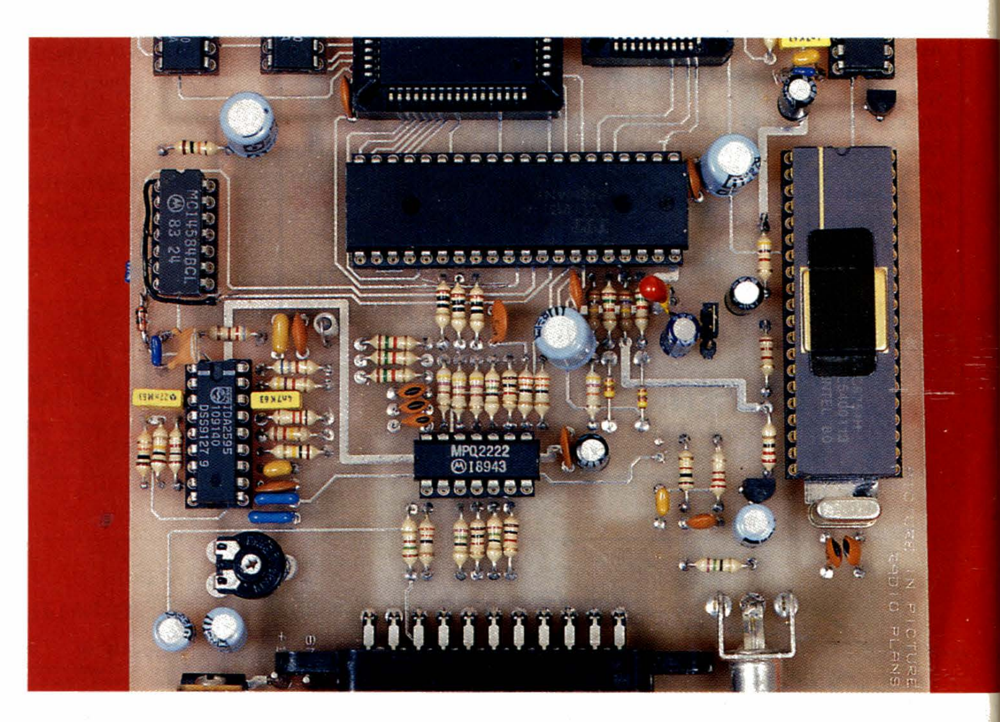


Interactif, DAQ Designer interroge l'utilisateur au travers d'une interface graphique très simple d'emploi. Il pose les questions nécessaires à la caractéristation de son application : types et nombres de signaux analogiques et numériques, types et nombres de capteurs, besoins en matière de conditionnement du signal, etc. Après analyse des réponses de l'utilisateur, DAQ Designer recommande des cartes d'acquisition de données enfichables, des produits de conditionnement du signal, des accessoires de câblage, et des logiciels pouvant être utilisés pour bâtir le système d'acquisition adéquat.
Une fois que l'utilisateur a fait ses choix parmi les éléments recommandés par DAQ Designer, le programme fournit une synthèse des caractéristiques de l'application, et la liste de produits sélectionnés. L'utilisateur peut sauvegarder les résultats sur disque et/ou les imprimer.
DAQ Designer nécessite un ordinateur équipé d'un écran VGA, avec au moins : la version 3.0 de DOS, un processeur 80286, et 640 Ko de mémoire RAM.

National Instruments
Centre d'affaires Paris-Nord -
BP 217 - 93153 Le Blanc-Mesnil
Tél. : (1) 48.65.33.70

Un ensemble P.I.P Application des circuits ITT DIGIT 2000 (2)

Nous avons abordé dans le précédent numéro l'étude des circuits ITT constituant le kit DIGIT 2000 qui permettent, entre autres applications, la réalisation de notre carte d'incrustation couleur. Dans ce deuxième volet, après un retour rapide sur le bus Intermetall — Imbus —, nous allons présenter le schéma de principe général ainsi que la réalisation pratique. Soulignons qu'hormis le Dump correspondant à notre paramètrage, le logiciel Inter-Exe disponible sur le serveur ERP ou par disquette vous permettra d'adapter cet ensemble à vos desiderata.



Imbus

Pour fonctionner correctement, ou conformément aux souhaits de l'utilisateur, les circuits doivent être configurés en suivant certaines règles : respecter le timing temporel d'introduction des données via le bus trois fils IMBus, et placer dans les registres internes 8 ou 16 bits les valeurs ad-hoc.

DEFINITION GENERALE DE L'IMBUS

Le bus intermetall est un bus trois fils reliant un microcontrôleur aux circuits Digit 2000, dans cette application : VSP 2860 et PIP 2250.

L'IMBus se compose de trois lignes :

- IMI : signal d'identification,
- IMC : signal d'horloge,
- IMD : signal de données.

Le diagramme des temps d'une transaction IMBus est représenté à la **figure 22**.

Sur les lignes IMI et IMC le signal transite de manière unidirectionnelle du microcontrôleur vers les processeurs. Sur la ligne de données IMD, les données transitent soit dans un sens soit dans l'autre. Le microcontrôleur peut donc envoyer des données ou lire des données. Dans tous les cas le microcontrôleur est maître et les processeurs du système Digit 2000 esclaves.

Le schéma de la figure 22 représente le diagramme des temps d'une transaction complète effectuée via l'IMBus. Au repos les trois lignes IMI, IMC et IMD sont à l'état haut.

Pour signaler le début du transfert, le maître, le microcontrôleur, fait passer la ligne identification à l'état bas.

Cette ligne reste à l'état bas pendant le transfert de l'adresse, sur IMD, cadencé par les fronts montants d'horloge IMC.

A la fin de ce transfert l'adresse est mémorisée dans les circuits esclaves et la comparaison s'ef-

fectue dès que le signal d'identification passe à l'état haut.
Les ordres d'écriture ou de lecture étant liés aux adresses, le circuit esclave concerné agit en conséquence.
Le contrôleur transmet ensuite 8 ou 16 coups d'horloge, en fonction du type de registre, et envoie ou lit 8 ou 16 bits de données.
Une brève impulsion sur le signal d'identification signale au circuit esclave que la transmission est terminée et cette impulsion mémorise le mot ou double mot dans le circuit esclave.

Adresse et sous-adresse

L'adresse est codée sur un mot de huit bits. Dans les tableaux précédemment évoqués, on remarque que le nombre de registres est assez important.
Si tous les registres de tous les circuits Digit 2000 étaient accessibles directement la présence simultanée de trois ou quatre circuits suffirait à saturer la totalité du champ d'adresse : 256.

Pour pallier cet inconvénient, le constructeur a prévu un accès indirect à chacun des registres.
Chaque circuit à trois adresses propres :
– écriture de sous-adresse (220 VSP 2860)
– écriture de données (222 VSP 2860)
– lecture de données (221 VSP 2860).
Pour écrire la valeur V dans le registre R, le microcontrôleur doit exécuter 2 transactions conformes au standard IMBus.
La première transaction consiste à faire passer R en signifiant que l'on veut passer une sous-adresse, soit sous forme IMBus : adresse : 220, data : R.
La deuxième transaction consiste à faire passer la valeur et signifier qu'il s'agit d'une données soit sous forme IMBus : adresse 222, data V.
Si l'on souhaitait lire une valeur interne seule la deuxième transaction serait modifiée : adresse 221, data V.

Lors de la description du VSP 2860 nous avons vu qu'un processeur rapide dénommé HSP devait être configuré par le contrôleur.
Celui-ci n'est pas accessible avec la procédure IMBus décrite précédemment. L'écriture ou la lecture nécessite le passage par un troisième niveau d'adressage : registre 34, 35, 36 et 37.
Le programme du microcontrôleur 87C51 que vous pourrez

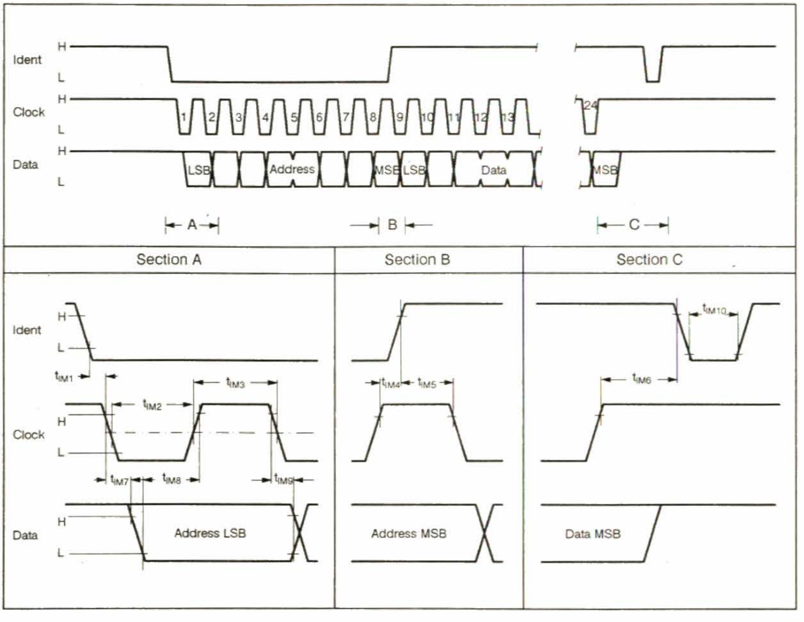
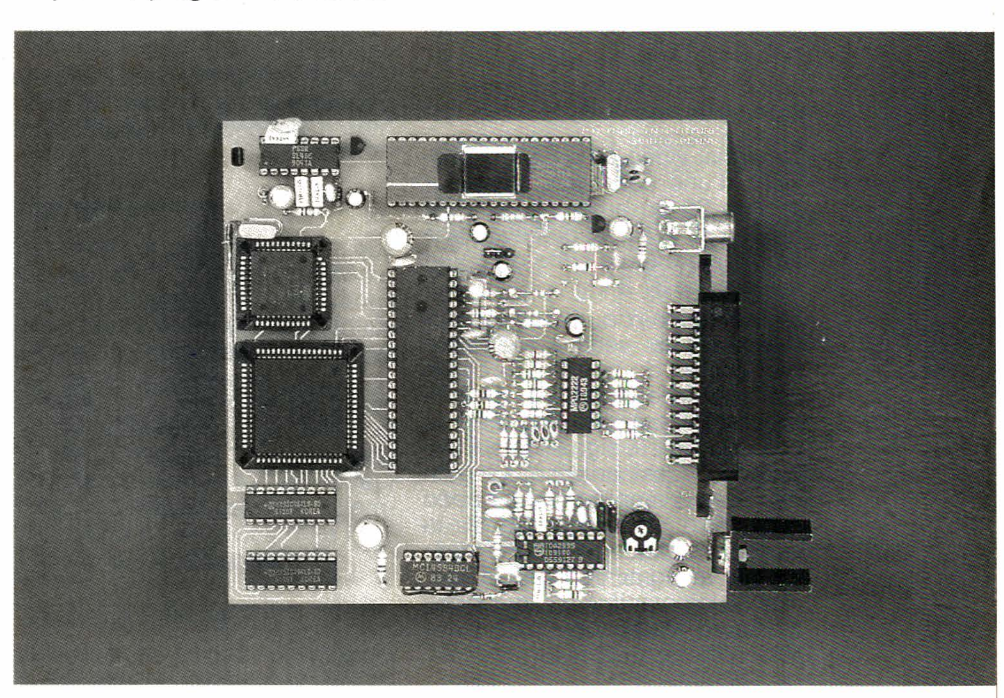


Figure 22

acquérir comporte bien entendu tous les sous-programmes destinés au paramétrage des circuits y compris les registres HSP nécessaires.
Ces procédures sont évidemment valables pour le PIP 2250 pour lequel les différentes adresses sont :
– écriture de sous-adresse 217,
– écriture de données 219,
– lecture de données 218.
Dans les deux cas, VSP et PIP les adresses sont exprimées en décimal.
Pour évaluer les performances des circuits et définir les valeurs à envoyer dans les nombreux registres, nous avons développé un logiciel PC écrit en C.
Si vous le souhaitez vous pourrez acquérir le programme exécutable transformant votre PC en outil de développement ou console dédiée IMBus.
Sur l'une des prises parallèles de votre PC, LPT1 ou LPT2, trois lignes sont affectées à IMI, IMC et IMD. Le protocole de transfert est respecté.
Ce principe est excellent car il permet un débuggage hardware rapide. Désolé pour cette accumulation de termes anglosaxons mais aucun autre n'est plus approprié.
La plupart des manipulations s'effectuent avec la souris, le chargement de valeur, incrémentation, décrémentation d'un registre est très simple. Cet outil pourra être utilisé, comme nous l'avons fait, pour l'évaluation des circuits.



Si l'objectif final est l'élaboration d'une carte d'incrustation avec un contrôleur d'un type autre que celui que nous avons utilisé : 87C51, il sera possible d'employer un émulateur pour le contrôleur choisi.
Cette solution est certainement moins souple et moins facile d'emploi car elle ne comprend pas l'interface graphique inclus dans notre programme.
Celui-ci comporte en outre une sauvegarde et restitution des valeurs de tous les paramètres. Après une séance d'essai, il est possible de sauvegarder la valeur de tous les paramètres en cours.
A la prochaine séance, on restitue toutes les valeurs, les essais continuent au stade où ils se situaient lors de la cloture de la première séance.
Finalement si l'objectif n'est qu'une réalisation de notre maquette, cette étape, bien que conseillée au moins par curiosité, pourra être totalement éliminée.
Il suffit alors de se procurer un contrôleur dûment programmé et le placer dans son support.

Le VCU 2133

Le schéma synoptique interne de ce circuit est représenté à la **figure 23**. Le VCU 2133 est destiné à la conversion analogique/numérique 7 bits d'un signal vidéo en bande de base. Après traitement externe ce même circuit convertit les signaux numériques Y, U, V en signaux analogiques R, V, B. Dans le concept Digit 2000 ITT, le signal à traiter peut être du type PAL, SECAM, NTSC ou D2MAC.

Convertisseur A/D

Le signal vidéo à numériser provient d'une des deux entrées - broche 35 ou broche 37 -. A la broche 35 le signal doit avoir une amplitude crête à crête de 1 V et à la broche 37 2 V.

La sélection de l'amplificateur vidéo mis en service s'effectue par le biais du bus IM. Si ce bit est à zéro le signal injecté à la broche 35 est analysé, dans le cas contraire le signal injecté à la broche 37 est analysé. Le gain des deux amplificateurs vidéo peut être doublé pendant la période d'effacement ligne définie par le signal injecté à la broche 36 de manière à obtenir une meilleure résolution de la salve d'identification couleur en PAL ou SECAM.

Le convertisseur A/D est du type flash, il comprend donc de nombreux comparateurs. Technologiquement un convertisseur flash 8 bits est deux fois plus difficile à fabriquer puisqu'il comprend deux fois plus de comparateurs. En principe pour une bonne définition un convertisseur 8 bits est nécessaire. Pour améliorer la résolution donnée par un convertisseur 7 bits une petite astuce est utilisée.
Une ligne sur deux, la tension de référence du convertisseur est modifiée d'une quantité correspondant à la moitié du bit le moins significatif. De cette manière un échantillon compris entre deux valeurs d'une échelle à sept bits se traduit une ligne sur deux en deux valeurs numériques n et n + 1. Ceci se traduisant sur l'écran par deux niveaux de gris différents. Les deux valeurs de gris sont moyennées par l'œil et produisent l'impression d'une échelle de gris due à un convertisseur 8 bits.
La résolution du convertisseur est de 1/2 LSB sur 8 bits. Les sept bits de sortie sont disponibles en parallèle et le codage du type Gray.

Inverseur de bruit

Si le niveau du signal vidéo atteint ou dépasse rapidement le maximum autorisé on peut considérer qu'il s'agit d'un parasite que l'inverseur de bruit remplace par un niveau de gris moyen. Cet inverseur de bruit peut être mis en ou hors service.

Convertisseur numérique-analogique pour la luminance

Après traitement, les signaux numériques luminance et différence de couleur retournent au convertisseur VCU 2133. La luminance, mot de huit bits, est appliquée aux entrées L0 à L7 - broches 17 à 10 -. Le convertisseur numérique/analogique est classique : emploi d'un réseau R-2R. Les divers interrupteurs du réseau sont actionnés au rythme de l'horloge appliquée à la broche 22. En D2MAC cette fréquence vaut 20, 25 MHz. En PAL ou SECAM la fréquence vaut 17,734 MHz.

Convertisseurs numériques pour les signaux différence de couleur

La bande des signaux différence de couleur étant plus étroite que la bande du signal de luminance, il est possible de transférer les signaux R-Y et B-Y non en parallèle mais en série, des processeurs vers les convertisseurs.
Ceci justifie une fois encore le format 4-1-1.

Matrice R, V, B et amplificateurs de sortie

Les signaux luminance et différence de couleur sont dématricés pour donner les signaux R, V, B. En outre la matrice reçoit une tension de décalage provenant d'un mot de huit bits. Cette tension de décalage agit sur la lumière de l'image. La plage d'ajustement de la lumière vaut la moitié de l'excursion acceptable en sortie du VCU 2133 : 6 V. La lumière est ajustée via l'IM-Bus.

Les trois amplificateurs de sortie R, V, B sont des convertisseurs d'impédance. L'excursion maximale de 6 V se répartit de la manière suivante : 3 V pour la tension de décalage - lumière -

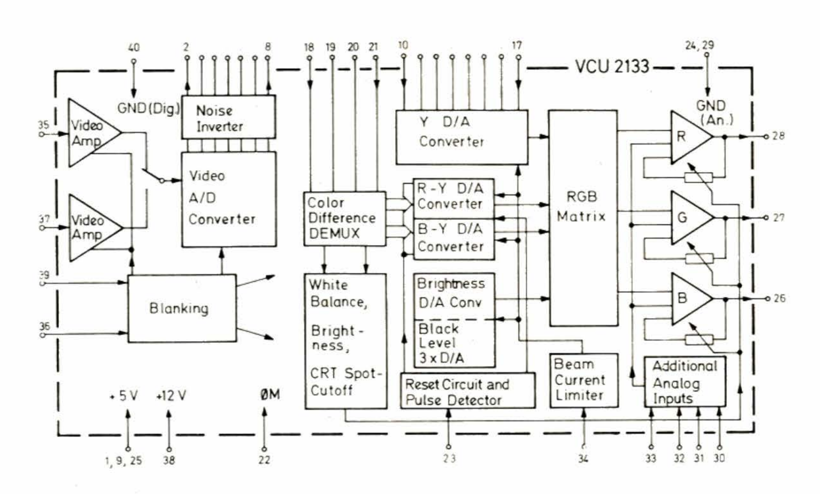


Figure 23

et 3 V pour le signal utile. Le courant de sortie maximal vaut 4 mA, ce qui signifie que l'interface de sortie devra comporter outre des filtres passe-bas, un buffer vidéo capable de débiter sur une charge de 75 Ohms si l'on souhaite attaquer les entrées R, V, B d'une prise PERITEL.

Limitation du courant de faisceau

Cette fonction n'est utilisée que lorsque les étages de sortie du VCU pilotent directement le tube cathodique.
La limitation du courant de faisceau s'effectue en modifiant contraste et lumière par action sur la tension de référence des trois convertisseurs D/A : luminance et différence de couleur.
La réduction est fonction de la tension appliquée à la broche 34 du VCU 2133. Pour une tension supérieure à 4 V, contraste et lumière ne sont pas affectés, pour une tension comprise entre 3 et 4 V le contraste décroit régulièrement. Pour une tension égale à 3 V le contraste est ramené à un niveau préprogrammé.
Lorsque, à la broche 34, la tension atteint 2 V la lumière passe à zéro et pour des tensions inférieures les sorties sont maintenues à l'infra noir.
Dans toute application vidéo, la broche 34 est mise à un potentiel supérieur à 4 V et la réduction du courant n'a aucun effet puisque cette entrée de retour de mesure de courant est à un niveau continu fixe.

SCHEMA DE PRINCIPE

Le schéma de principe du système d'incrustation d'image dans l'image est représenté à la **figure 24**.
Du signal vidéocomposite principal, on extrait, grâce à un TDA 2595, les deux signaux de synchronisation H et V.
La gestion des deux circuits est confiée à un microcontrôleur 87C51 UV ou OTP. Trois bits d'un port d'entrée/sortie banalisé sont affectés arbitrairement à l'IMBus : IMClock, IMIdentification et IMDonnées.
Le microcontrôleur gère aussi la remise à zéro des circuits Digit 2000.
Le programme inscrit en EPROM traite les informations d'entrée INTO comme les informations de contrôle issues du récepteur de télécommande infrarouge SL 486.
La télécommande prévue pour cette fonction est du type IR 5904. Aucun accès clavier n'est prévu. Cette télécommande permet seulement le déplacement horizontal et vertical de l'image incrustée.

REALISATION PRATIQUE

L'ensemble des composants du schéma de principe de la figure 24 prend place sur une carte imprimée double face de faibles dimensions.
Le tracé des pistes côté soudure est représenté à la **figure 25** et côté composants à la **figure 26**.
L'implantation correspondante est donnée à la **figure 27**.
Tous les circuits intégrés sont montés sur support. On veillera à l'orientation des supports et des circuits VSP 2860 et PIP 2250.
Le circuit VSP 2860 en boîtier plastique PLCC 44 broches et le PIP 2250 en boîtier plastique PLCC 68 broches nécessitent des supports spéciaux.
L'équipement de la carte ne devrait pas poser de problème et les circuits intégrés spécifiques ITT devraient être distribués par ITT Multicomposants aux ULIS.

Mise sous tension

La carte est équipée, sans son microcontrôleur et peut être mise sous tension. Le module PIP est relié à un téléviseur via un cordon Péritel/Péritel croisé.
On règle premièrement le potentiomètre de 4,7 kΩ du TDA 2595 pour obtenir aux sorties 4 et 9 les synchronisations ligne et trame.
A ce stade de la réalisation deux voies s'ouvrent à vous, la première : mise en place immédiate du contrôleur programmé et fin de la réalisation où la seconde, contrôle du bon fonctionnement du système par un PC émulateur.

Première solution

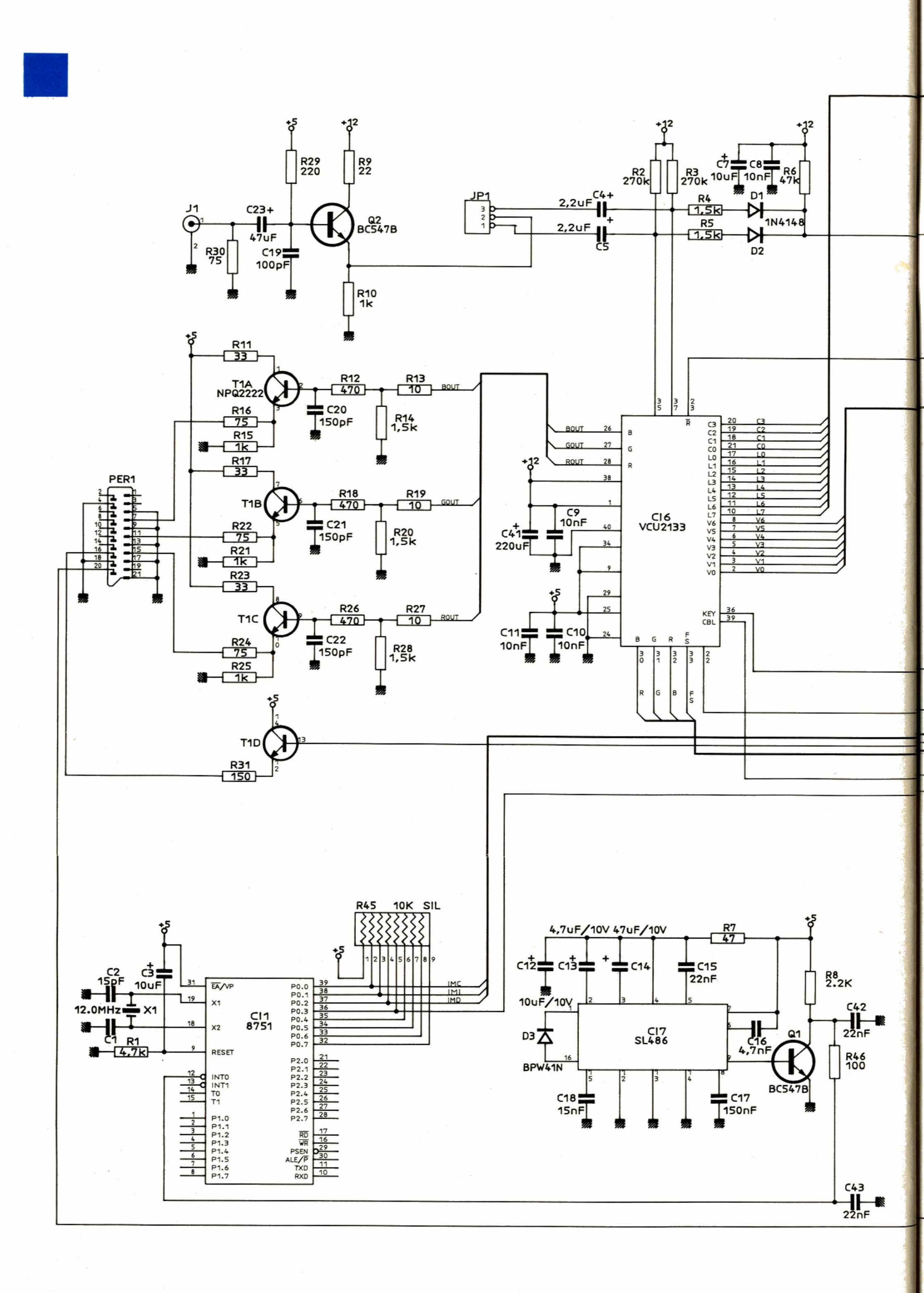
Si votre but profond n'était que la réalisation et surtout l'utilisation du module PIP, vous vous procurez simplement un contrôleur dûment programmé et l'insérez dans son support.
Il suffit alors d'injecter sur J1 un signal vidéocomposite PAL, et ajuster C40 pour avoir un bon fonctionnement du décodeur PAL. Le module est prêt à l'emploi, vous pourrez contrôler l'action de la télécommande IR.

Deuxième solution

Si vous souhaitez développer votre propre code pour 87C51 ou tout autre microcontrôleur, cette deuxième solution devrait vous faciliter la tâche.
Une des sorties LPT1 ou LPT2 de votre PC sera utilisée comme liaison IMBus avec les circuits du module PIP.
Le schéma de la **figure 28** donne l'affectation des broches du connecteur LPT.
Cette figure montre qu'il est assez simple de réaliser un adaptateur PC/module PIP.
Le soft PC que vous pourrez vous procurer auprès de la rédaction d'ERP comporte trois fichiers :
- PIP.DAT
- INTER.EXE
- POLICE.VGA.

Une seule page écran permet le paramétrage des trois circuits du module PIP. Une souris est indispensable pour cliquer sur le paramètre devant être modifié.

Grâce à quatre circuits de la famille Digit 2000 on réalise simplement un système d'incrustation. Les deux synoptiques des **figures 29** et **30** donnent une idée de ce que pourrait être un générateur de mosaïque.
Pour cette mosaïque nous avons deux approches différentes, une approche analogique à la figure 29 et numérique à la figure 30.
La figure 29 regroupe, pour l'exemple, quatre circuits d'incrustation identiques à celui décrit dans les pages précédentes.
Chacun de ces circuits reçoit un signal de synchronisation ligne, un signal de synchronisation trame et un signal d'horloge à 17,734 MHz. Chacun des systèmes délivre les signaux R, V, B relatifs à l'image miniature.
Grâce à un multiplexeur 1 parmi 4 on élabore les signaux R, V, B analogiques de l'image principale. En l'occurence celle-ci est constituée de 4 images miniatures. Le fond n'est pas défini et est donc noir.
Aux signaux R, V, B d'entrée du codeur PAL on peut ajouter les signaux R, V, B en provenance d'un générateur de caractères. Ce générateur résoud simplement le problème de l'identification des sources.
Après codage on dispose d'un signal vidéocomposite pouvant être enregistré ou distribué.
La figure 29 montre que cette mosaïque peut être réalisée sans sortir du concept Digit 2000 ITT.
Le PIP fournit l'image miniature sur le Bus 12 bits Y, U, V. Ces informations sont multiplexées et envoyées au codeur multistandard de la même famille : MSE 3000.



+5
+12
R29
220
R9
22
J1
C23+
47uF
Q2
BC547B
R30
75
C19
100pF
R10
1k
JP1
2,2uF
C4+
C5
R2
270k
R3
270k
C7
10uF
C8
10nF
R6
47k
R4
1,5k
R5
1,5k
D1
1N4148
D2
R11
33
T1A
NPQ2222
R12
470
R13
10
BOUT
C20
150pF
R14
1,5k
R16
75
R15
1k
R17
33
PER1
T1B
R18
470
R19
10
GOUT
C21
150pF
R20
1,5k
R22
75
R21
1k
R23
33
T1C
R26
470
R27
10
ROUT
C22
150pF
R28
1,5k
R24
75
R25
1k
T1D
R31
150
C9
10nF
C41
220uF
C11
10nF
C10
10nF
CI6
VCU2133
KEY
CBL
R45 10K SIL
C2
15pF
C3
10uF
12.0MHz
X1
C1
R1
4,7k
CI1
8751
EA/VP
X1
X2
RESET
INTO
INT1
T0
T1
P1.0
P1.1
P1.2
P1.3
P1.4
P1.5
P1.6
P1.7
P0.0
P0.1
P0.2
P0.3
P0.4
P0.5
P0.6
P0.7
P2.0
P2.1
P2.2
P2.3
P2.4
P2.5
P2.6
P2.7
RD
WR
PSEN
ALE/P
TXD
RXD
IMC
IMI
IMD
4,7uF/10V 47uF/10V
R7
47
C12
C13
C14
C15
22nF
10uF/10V
D3
BPW41N
CI7
SL486
C16
4,7nF
Q1
BC547B
R8
2.2K
C42
22nF
R46
100
C18
15nF
C17
150nF
C43
22nF

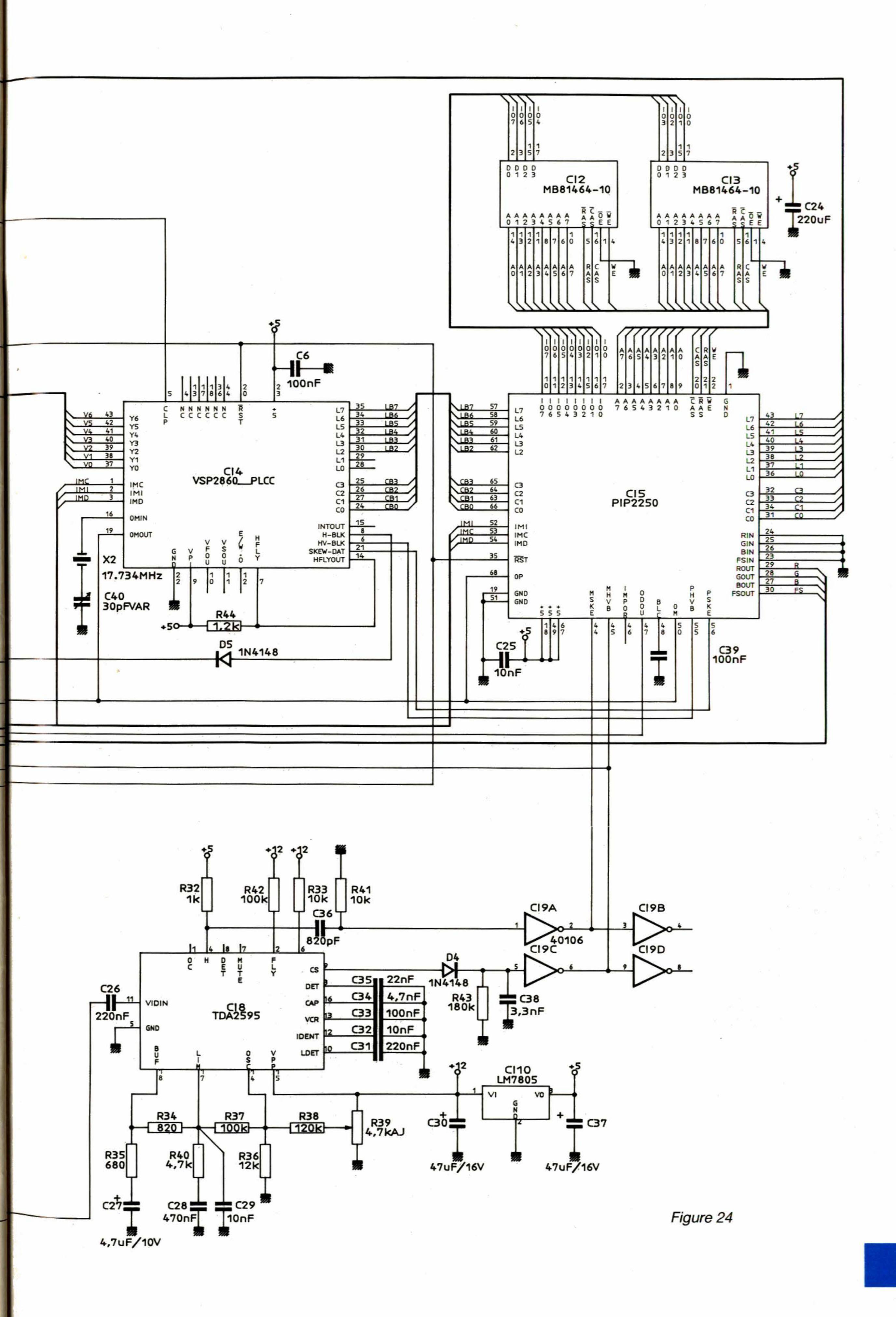

CI2
MB81464-10
CI3
MB81464-10
C24
220uF
C6
100nF
CI4
VSP2860 PLCC
CI5
PIP2250
X2
17.734MHz
C40
30pFVAR
R44
1,2k
D5
1N4148
C25
10nF
C39
100nF
R32
1k
R42
100k
R33
10k
R41
10k
C36
820pF
CI9A
CI9B
CI9C
CI9D
40106
D4
1N4148
R43
180k
C38
3,3nF
C26
220nF
CI8
TDA2595
C35
22nF
C34
4,7nF
C33
100nF
C32
10nF
C31
220nF
CI10
LM7805
C30
47uF/16V
C37
47uF/16V
R34
820
R37
100k
R38
120k
R39
4,7kAJ
R35
680
R40
4,7k
R36
12k
C27
4,7uF/10V
C28
470nF
C29
10nF

Figure 24

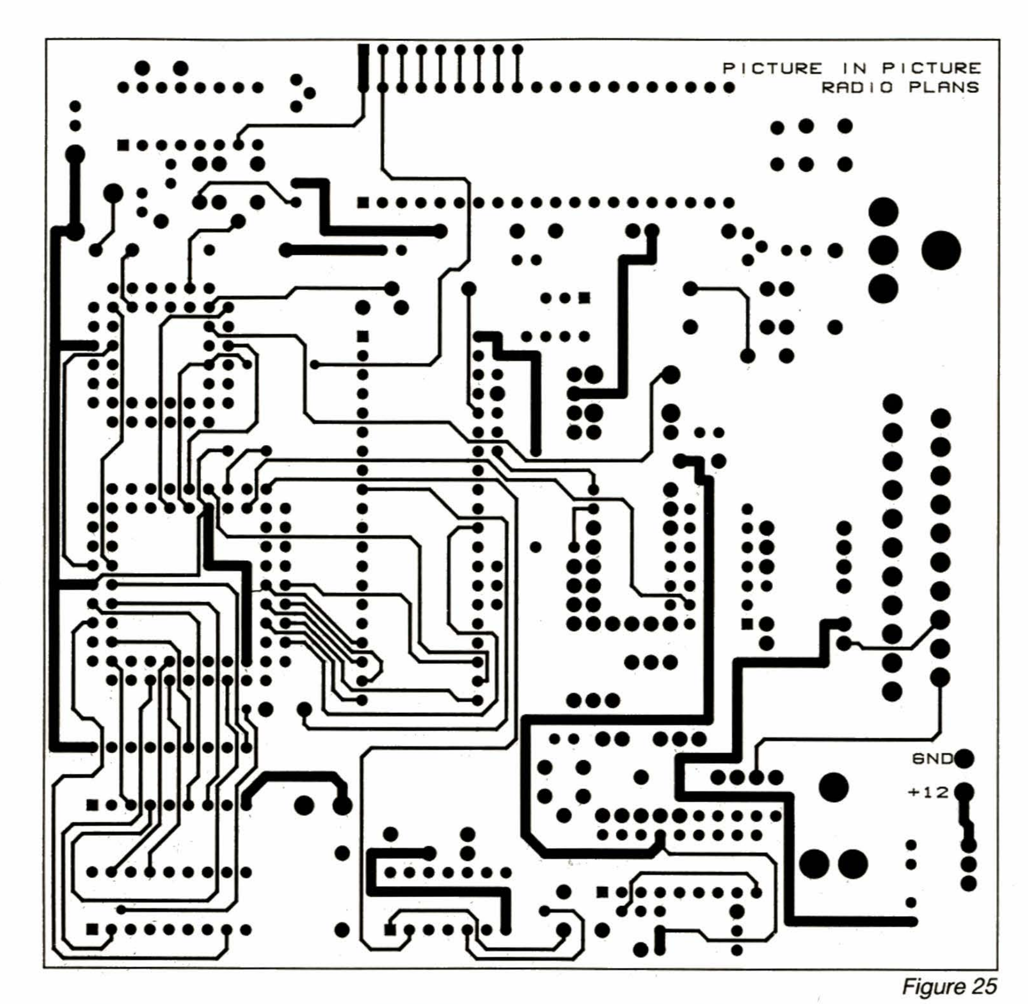


Figure 25

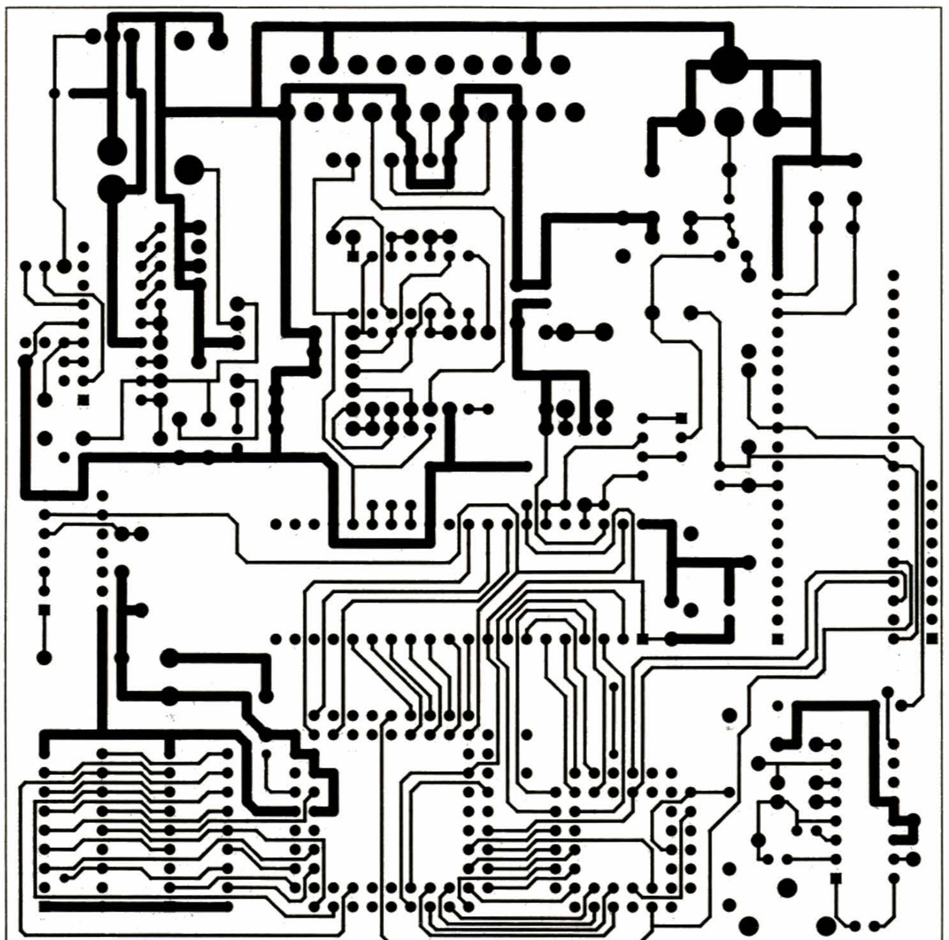


Figure 26

Par rapport au synoptique de la figure, cette solution est évidemment plus simple car le passage par les signaux R, V, B analogiques est évité.
L'inconvénient majeur de cette solution est la difficulté d'insertion de caractères. Dans ce nouveau cas, le codeur ne comportant pas d'entrées R, V, B externes, les titres devront obligatoirement être ajoutés au niveau du bus 12 bits Y, U, V.
Cette réalisation, simple puisque sans réglage, d'un faible coût devrait, il nous semble, intéresser la majorité d'entre vous.
Le module PIP autonome est un bon exemple d'application autour des circuits Digit 2000 ITT.
Dans quelques mois nous reviendrons sur ce concept avec une nouvelle application vidéo : le transcodage.
Quatre circuits de cette famille : VCU 2136, VSP 2860, SPU 2243 et MSE 3000 permettent la réalisation d'un transcodeur multinorme PAL/SECAM/SVHS vers PAL/SECAM/SVHS et ceci sans aucun réglage ni bobine...

Gilles De DIEULEVEULT
François De DIEULEVEULT

NDLR : Rappelons que le DUMP ainsi que le logiciel INTER.EXE. permettant de réaliser ses propres DUMPs sont disponibles sur le 36-15 ERP mais aussi contre l'envoi d'une disquette 360 k formatée avec port de retour joint.



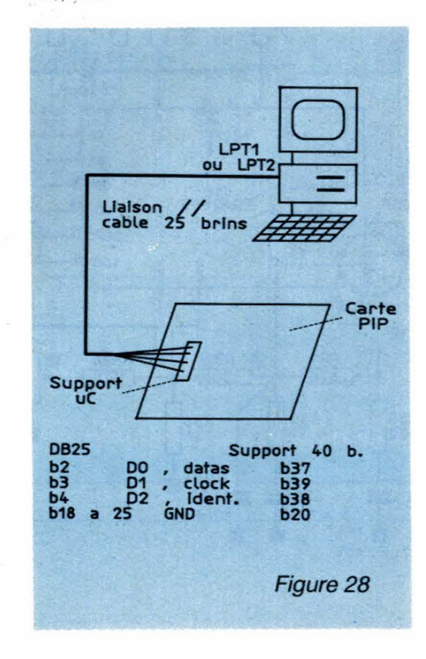


Figure 28

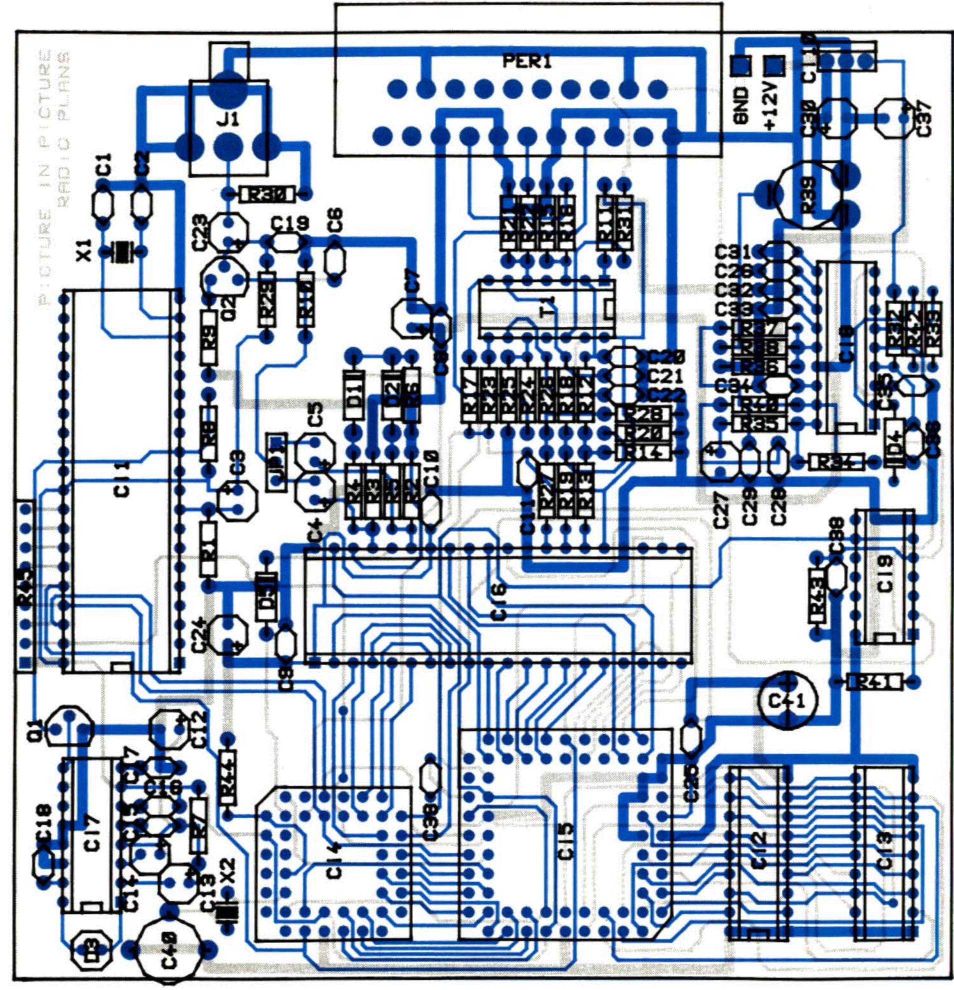


Figure 27

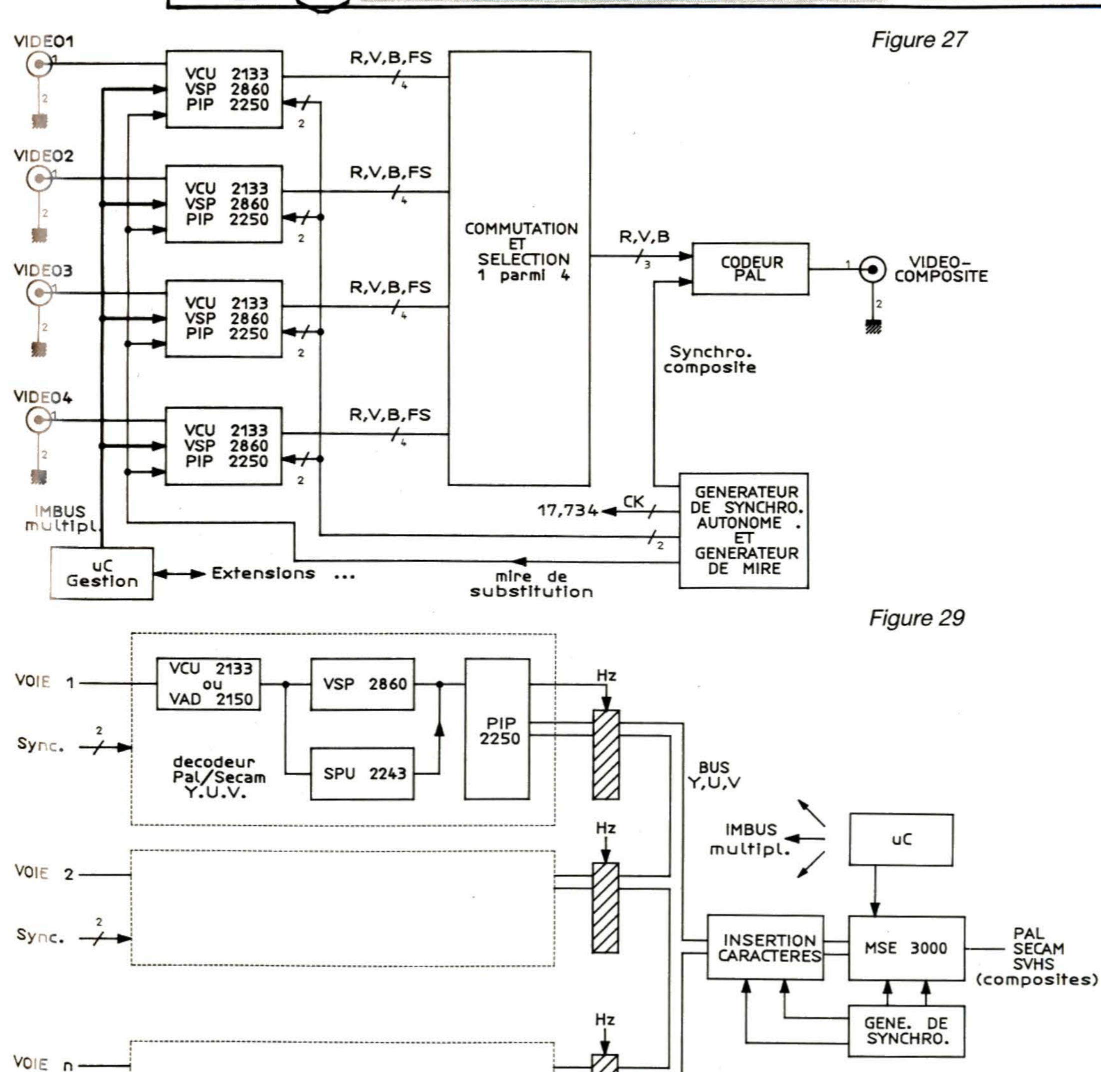


Figure 29

Figure 30

Nomenclature

Résistances

R_1, R_{40} : 4,7 kΩ
R_2, R_3 : 270 kΩ
R_4, R_5 : 1,5 kΩ
R_6 : 47 kΩ
R_7 : 47 Ω
R_8 : 2,2 kΩ
R_9 : 22 Ω
R_{10}, R_{15}, R_{21}, R_{25}, R_{32} : 1 kΩ
R_{11}, R_{17}, R_{23} : 33 Ω
R_{12}, R_{18}, R_{26} : 470 Ω
R_{13}, R_{19}, R_{27} : 10 Ω
R_{14}, R_{20}, R_{28} : 1,5 kΩ
R_{16}, R_{22}, R_{24}, R_{30} : 75 Ω
R_{29} : 220 Ω
R_{31} : 150 Ω
R_{33}, R_{41} : 10 kΩ
R_{34} : 820 Ω
R_{35} : 680 Ω
R_{36} : 12 kΩ
R_{37}, R_{42} : 100 kΩ
R_{38} : 120 kΩ
R_{39} : 4,7 kΩ
R_{43} : 180 kΩ
R_{44} : 1,2 kΩ
R_{45} : réseau 10 kΩ SIL

Condensateurs

C_1, C_2 : 15 pF
C_3, C_7 : 10 µF
C_4, C_5 : 2,2 µF
C_6, C_{33}, C_{39} : 100 nF
C_8, C_9, C_{10}, C_{11}, C_{25}, C_{29}, C_{32} : 10 nF
C_{12} : 10 µF/10 V
C_{13}, C_{27} : 4,7 µF/10 V
C_{14} : 47 µF/10 V
C_{15}, C_{35} : 22 nF
C_{16}, C_{34} : 4,7 nF
C_{17} : 150 nF
C_{18} : 15 nF
C_{19} : 100 pF
C_{20}, C_{21}, C_{22} : 150 pF
C_{23} : 47 µF
C_{24}, C_{41} : 220 µF
C_{26}, C_{31} : 220 nF
C_{28} : 0,47 µF
C_{30}, C_{37} : 47 µF/16 V
C_{36} : 820 pF
C_{38} : 3,3 nF
C_{40} : 30 pF

Semi-conducteurs

D_1, D_2, D_4, D_5 : 1N 4148
D_3 : BPW41N
Q_1, Q_2 : BC547B
T_1 : NPQ 2222

Circuits intégrés

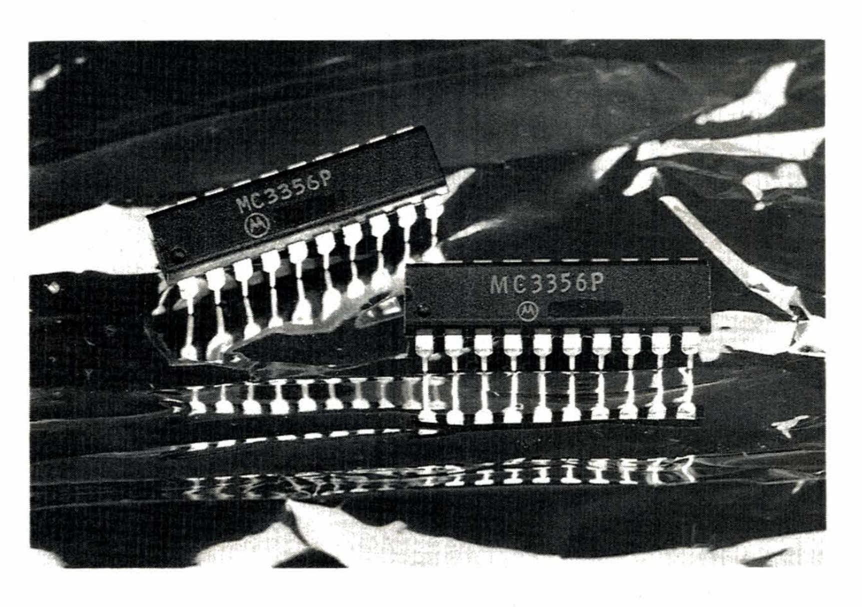
U_1 : 8751
U_2, U_3 : MB81464 DRAM 64 K x 4 80 ns
U_4 : VSP 2860 PLCC ITT
U_5 : PIP 2250 ITT
U_6 : VCU 2133 ITT
U_7 : SL486 PLESSEY
U_8 : TDA 2595 PHILIPS
U_9 : 40106
U_{10} : LM 7805

Divers

J_1 : BNC
JP_1 : Connecteur 3 points
PER_1 : PERITEL EMB
X_1 : quartz 12,0 MHz
X_2 : quartz 17,734 MHz

Le MC 3356, MOTOROLA

Bien que pas vraiment récent, ce circuit intégré a retenu notre attention pour ses caractéristiques intéressantes. En effet, il a sa place dans les applications de transmission numérique jusqu'à 500 kbauds, ou analogiques jusqu'à 250 kHz. Sa conception interne autorise la mise en œuvre de récepteurs à simple changement de fréquence jusqu'à 200 MHz ou double avec des étages FI qui montent à 50 MHz. Il dispose d'un circuit de détection de niveau RSSI qui présente une dynamique de 80 dB, allant de 20 µV à 200 mV. Pour un rapport signal sur bruit de 50 dB à la sortie, la sensibilité vaut 60 µV.



Le MC 3356 est disponible en deux boîtiers différents, CMS en PLCC 775-02 et DIL 738-03, tous deux à 20 broches haute intégration dont le schéma synoptique s'articule autour des éléments décrits à la **figure 1**. Il nécessite deux tensions d'alimentation dûment découplées par des condensateurs au tantale de préférence, pour une réduction de l'effet inductif aux plus basses fréquences. Les broches 6 et 10 correspondent aux sections FI et démodulation qui demandent une valeur d'alimentation entre 6 et 9 volts. La broche 4 s'occupe des sections oscillateur et mélangeur, la tension peut aller jusqu'à 12 volts si l'on souhaite travailler à 200 MHz, le gain de conversion vaut 2. La consommation maximale atteint 30 milliAmpères.

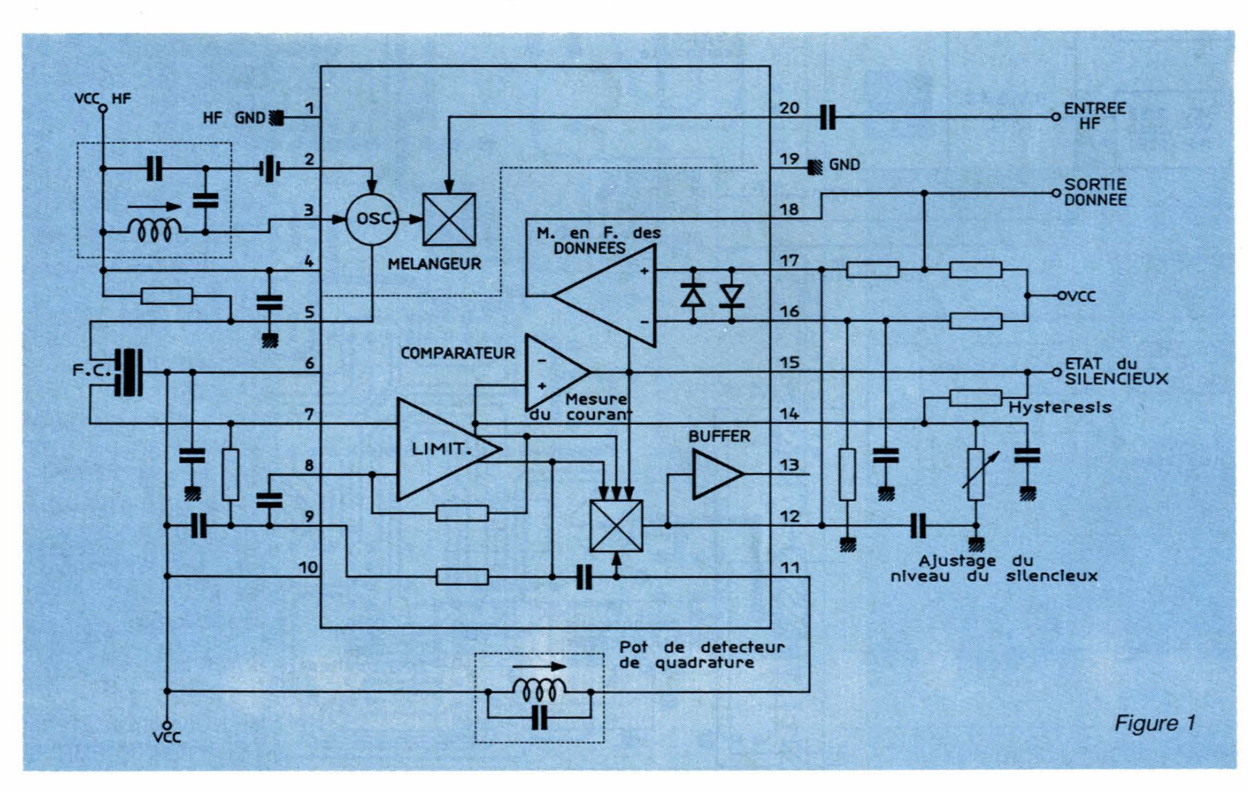


Figure 1

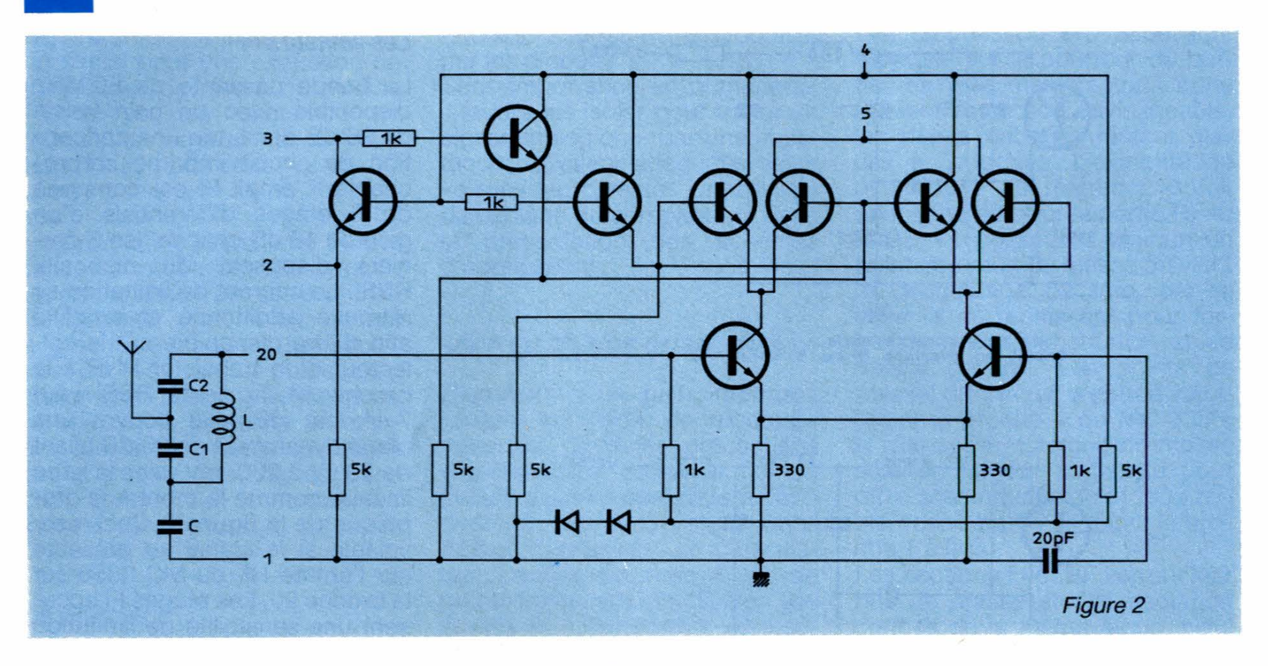


Figure 2

L'étage d'entrée

Comme le montre le schéma de la **figure 2**, l'entrée HF se trouve sur la broche 20 du circuit, le mélangeur se présente sous la forme d'un amplificateur différentiel dont la conception rappelle celle des structures de GILBERT. Elle offre une excellente réjection des signaux d'entrée et d'oscillateur local. A 100 MHz, l'impédance vaut 260 Ohms en parallèle sur une capacité de 5 pF. Cet étage peut commencer à travailler avec une amplitude de 10 µV, on note alors une dégradation du rapport signal sur bruit qui descend à 25 dB. Il reste toujours possible d'utiliser un préamplificateur avant le mélangeur, un gain de 20 dB permettra d'obtenir une sensibilité voisine de 3 µV. A ce moment, la dynamique d'utilisation du récepteur s'en ressent, il s'agira de choisir le meilleur compromis par rapport à l'application envisagée. Les meilleures performances du MC 3356 se situent lorsque la tension VccHF, sur la broche 4, se situe entre 10 et 12 volts. Un transistor interne assure la fonction d'oscillateur en base commune.

L'émetteur de ce transistor se trouve sur la broche 2 et son collecteur sur la broche 3. La sortie du mélangeur dirige le signal à fréquence intermédiaire sur un filtre de bande. MOTOROLA suggère l'emploi de filtres céramique dont les impédances d'entrée-sortie valent 330 Ohms. Dans certaines applications il sera nécessaire d'utiliser la sortie du mélangeur jusqu'à 130 MHz, ceci est rendu possible par l'emploi d'un filtre LC entre les broches 4 et 5.

Le couplage d'antenne

Alors que le simulateur PSPICE apporte une solution rapide au calcul des éléments du filtre d'entrée, comme le montre l'annexe en fin d'article, nous avons déterminé ceux-ci à l'aide de formules simples.

On suppose que le récepteur doit fonctionner sur 100 MHz, la bobine choisie possède une inductance de 100 nH avec un facteur de surtension de 70, elle sera placée aux bornes d'un réseau série composé par deux capacités réalisant l'accord, revoir la **figure 2**. On s'arrange pour que l'impédance d'entrée soit de 50 Ohms. La première étape consiste à trouver la réactance de la bobine, $Xp = 2\pi \cdot f_0 \cdot L$ = 63 Ohms à 100 MHz. La résistance vaut alors $Rp = Qp \times Xp$ = 4 400 Ohms.

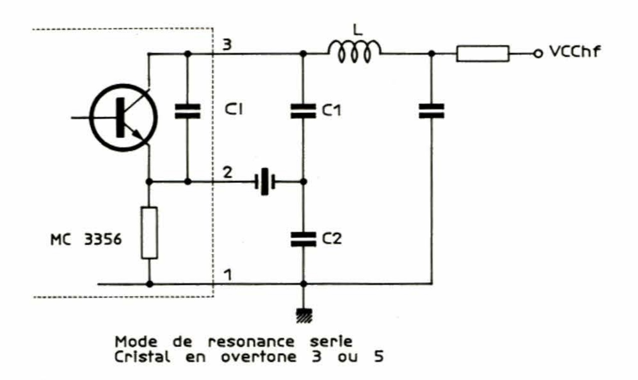
Le facteur de surtension en charge vaut $Qc = Rtot / Xp$ avec $Rtot = Re \times Rp / Re + Rp$. La valeur obtenue de $Qc = 4$ met en évidence la largeur de bande élevée, due à la présence de la faible résistance d'entrée du MC 3356. Les capacités C_1 et C_2 vont réaliser l'adaptation d'impédance avec l'antenne et, bien sûr l'accord sur la fréquence centrale. Pour avoir l'adaptation il faut satisfaire la condition $(C_1/C_2) = \sqrt{(Re/Rant)} - 1$, avec Re = 260 Ohms et Rant = 50 Ohms. Après calculs, $C_1/C_2 = 1,3$. A la résonance, la réactance capacitive doit être égale à la réactance inductive. La valeur du groupement série sur l'inductance vaut alors $C_{tot} = 1/(Xp \times 2 \times PI \times f_0)$ = 25 pF, avec $Ctot = C_1 \times C_2 / (C_1 + C_2)$ il vient : $C_1 = 2,3 \times Ctot$ et $C_2 = C_1/1,3$. Dans la pratique on prend C_1 = 56 pF et C_2 = 39 + 4,7 pF en parallèle. Par acquit de conscience on pourra vérifier la fréquence de résonance à l'aide de la formule de Thomson. Il faut maintenant prendre en compte la capacité de 5 pF imputable au MC 3356 qui vient en parallèle sur le réseau accordé. Il faut la soustraire de Ctot, tel que Ce = Ctot − 5 pF.

Les nouvelles valeurs correspondent à C_1 = 47 pF et C_2 = 33 pF + 2,2 pF en parallèle. La bande passante à − 3 dB vaut f_0 / Qc soit 12 MHz de part et d'autre avec 3 dB de perte d'insertion ! Pour améliorer les choses, on peut doter le montage d'un préamplificateur sélectif pourvu d'un transistor à effet de champ, un modèle J310 convient très bien. Sur sa porte on place un double circuit accordé qui sera moins amorti du fait de la résistance d'entrée très élevée du FET. Par jeu nous avons augmenté le rapport C_1 / C_2 jusqu'à 4,2, pour une même impédance d'antenne, la bande passante s'élargit confortablement. A titre indicatif, on se reportera aux différentes courbes présentées en annexe venant d'un fichier PSPICE.

L'oscillateur interne

Il peut fonctionner avec des quartz ou un simple réseau accordé. La réaction s'opère entre collecteur et émetteur du transistor interne, l'oscillation reste contrôlée par le rapport des capacités tel que C_2 = (1,5 à 2,5) × C_1, il détermine l'entretien des oscillations. Le schéma de

principe présenté à la **figure 3** propose la configuration de base ; le quartz est un modèle en résonance série qui travaille sur son harmonique 3, 5 ou 7. Sur cette figure, la capacité “CI” correspond à l’élément interne du MC 3356, sa valeur va de 3 à 10 pF selon les lots de fabrication. Pour réduire son influence on peut placer une résistance de 100 Ohms entre le collecteur et le point commun de C_1/L ce qui de toute façon est conseillé pour stabiliser le montage. Sur le deuxième schéma on voit cette modification avec en plus une résistance de 1 800 Ohms entre la broche 2 et la masse, elle augmente le courant collecteur et assure une puissance d’oscillateur plus importante. Ceci reste valable pour des quartz travaillant en harmonique 5 ou 7, où le niveau de sortie serait insuffisant pour garantir un gain de conversion de 2.



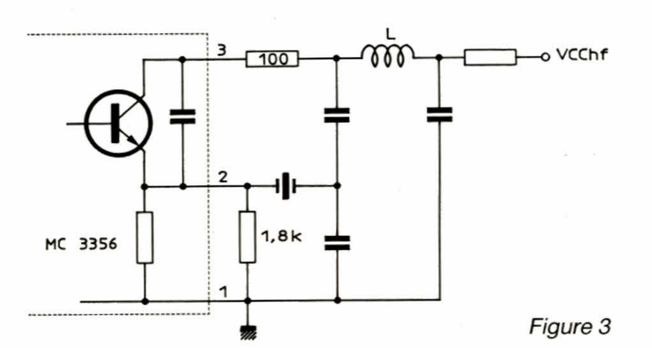


Figure 3

Les limiteurs FI

La bande passante de 50 MHz disponible avec un gain voisin de 96 dB demande une conception de circuit imprimé irréprochable. L’ampli FI est composé de 6 étages différentiels d’un gain de 16 dB chacun, les 5 derniers disposent du dispositif RSSI. Le courant de limitation se retrouve additionné et amplifié afin d’être disponible sur le collecteur d’un transistor PNP à la broche 14, la pente RSSI vaut 7 µA par dB, elle couvre une plage dynamique de 80 dB allant de 20 µV à 200 mV dans la zone linéaire comme le montre le graphique de la **figure 4**. Ceci reste valable si le signal se présente sur l’entrée HF du MC 3356 sur la broche 20. Les étages FI accusent une sensibilité de limitation de 50 µvolts. Les 6 étages limiteurs sont couplés en courant continu jusqu’au démodulateur.
Le MC 3356 permet d’employer des fréquences intermédiaires très élevées, les impédances de sortie du mélangeur et d’entrée des étages FI autorisent l’emploi de filtres céramique à 10,7 MHz, 6,5 MHz ou 5,5 MHz ou LC.
La sensibilité de limitation de la FI attaquée seule sur la broche 7 vaut 50 µV.

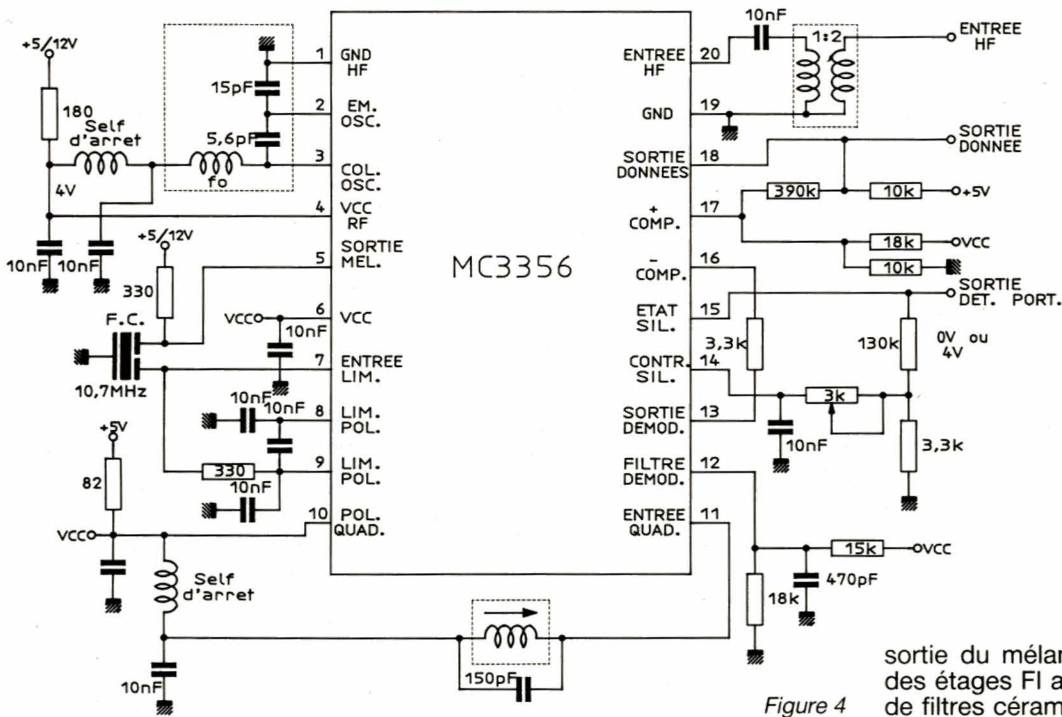


Figure 4

Le démodulateur FM

Il se compose à l’identique des circuits MOTOROLA déjà décrits mais supporte des fréquences à démoduler jusqu’à 50 MHz, il restitue les signaux BF avec une bande passante de 250 kHz. Cette bande passante permet la démodulation de signaux numériques jusqu’à 500 kbits/s.

Le circuit fournit jusqu'à 500 mV de signal pour une excursion de fréquence de plus ou moins 75 kHz et une fréquence de modulation équivalente à 1 000 Hertz sur sa broche 12 et 13 quand il faut une sortie "bufférisée".
La broche 11 reçoit une inductance accordée par une capacité, en parallèle il est souvent utile de disposer une résistance qui augmente la plage de démodulation en réduisant le facteur de surtension. Il paraît dérisoire de revenir sur la méthode d'obtention de la résistance que nous avons moulte fois décrite dans ces lignes. Le condensateur interne de déphasage possède une capacité de 5 pF.

Le silencieux

Il se compose d'un comparateur de niveau excité par la tension créée par le courant RSSI aux bornes d'une résistance. Le circuit bascule lorsqu'un potentiel de 800 mV se présente sur son entrée positive. Le seuil intrinsèque du comparateur étant invariable, il faut donc modifier la résistance d'attaque pour provoquer l'action du "squelch" à un niveau d'entrée HF différent. La sortie du silencieux rejoint d'un côté un transistor de commutation mettant hors/en circuit les sorties aux broches 12 et 13. De l'autre côté il va au circuit trigger disponible aux broches 16, 17 et 18. Ainsi lorsque le silencieux entre en action, niveau d'entrée trop faible, il conduit à la saturation les transistors aux broches 12 et 13 pour porter le niveau de sortie à celui de la masse. La broche 15 voit donc un niveau bas apparaître dans ce cas, la présence d'une porteuse porte cet accès à une tension presque identique à Vcc.

Le comparateur

Commandé par le silencieux, il rentre en action lorsque le niveau d'entrée devient suffisant. Son rôle consiste à mettre en forme les données numériques démodulées. Ses entrées flottantes restent protégées par deux diodes tête-bêche ; il est possible d'attaquer l'une ou l'autre entrée, inverseuse ou non, suivant le mode de conversion choisi (supradyne ou infradyne) ou le module qui suit le récepteur. Les fronts seront ainsi en lancée négative ou positive selon les besoins. En réunissant directement les broches 12 et 13 aux entrées du comparateur il apparaît les distorsions provoquées par les diodes.
Sans importance en numérique, il faudra les isoler pour traiter un signal analogique, par une résistance de valeur adéquate. Nous verrons les diverses possibilités de cablage du comparateur dans la présentation des schémas d'applications.

MISE EN ŒUVRE DU MC 3356

Disposant d'un gain important jusqu'à 200 MHz, ce récepteur nécessite certaines précautions d'implantation. La conception du circuit imprimé contribue à l'élaboration d'une structure sans défaut ; instabilité de l'oscillateur, retour de masse ou de points froids, auto-oscillation des étages FI. L'élimination de couplages d'entrée/sortie s'obtient alors en adoptant des inductances blindées. L'utilisation de perles ferrite sur le parcours de la ligne d'alimentation parfaitement découplée pour chaque étage prend ici toute sa valeur.
S'il devient possible d'éliminer les blindages, il n'en demeure pas moins indispensable de réaliser une implantation sur un circuit imprimé double face, avec un plan de masse sur la face supérieure côté composants. Les broches du MC 3356 sont réparties sur le pourtour du boîtier de telle manière qu'aucune interaction ne soit envisageable, les points critiques restent placés à l'opposée. L'alimentation et le point de référence 0 Volt sont séparés pour la partie RF et FI/BF, ils possèdent chacun un point commun de masse où vient le découplage de Vcc, par la suite ils se réunissent pour former le bus d'alimentation et de masse général. Les points de masse du circuit d'entrée rejoignent la broche 1 du MC 3356, la masse RF. La ligne de masse sera de largeur suffisante pour offrir une inductance la plus faible possible afin de réunir la broche 1 à la 9.
Les sections FI et BF verront leur bus de masse et d'alimentation conçus de la même façon, celui de masse ira ensuite sur la broche 19 par un chemin court et épais.

Schémas d'applications

Le dessin de la **figure 4** donne l'application du MC 3356 dans un récepteur pour données numériques à large bande. L'oscillateur local délivre une fréquence inférieure de 10,7 MHz par rapport à celle de réception, ce qui pour un signal FSK

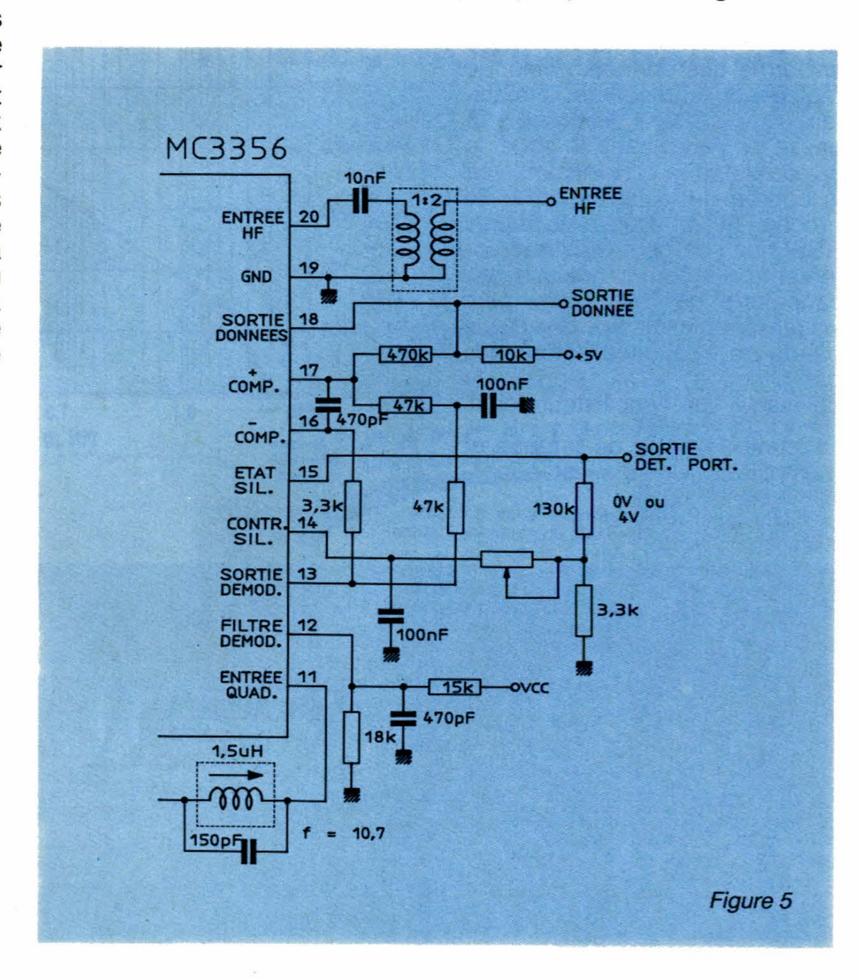


Figure 5

demande de rentrer sur la voie inverseuse de l'amplificateur opérationnel, broche 16.

L'hystérésis est donnée par la résistance de 390 kΩ, la référence DC sur la voie directe, broche 17 reste sensiblement la même que la tension continue disponible à la sortie du démodulateur, broche 13, mais lorsqu'aucun signal ne se présente sur l'entrée RF, les résistances de 18 kΩ et 10 kΩ se chargent de cette tâche.

Sur la **figure 5**, la tension de référence provient directement du démodulateur au travers d'un filtre passe-bas dont la constante de temps est très basse vis-à-vis de la fréquence la plus basse du train de données. L'intérêt de ce circuit consiste à compenser les dérives soit de l'oscillateur local soit de l'accord du réseau de quadrature.

La **figure 6** montre que l'on peut rentrer sur la voie directe, broche 17 du comparateur. Dans ce cas la fréquence de l'oscillateur local est supérieure à celle de réception. Dans tous les cas, et particulièrement en signaux analogiques, la sortie s'obtient sur la broche 12 du MC 3356 où un niveau de sortie de 500 mVcàc est présent. Le circuit du silencieux disponible sur la broche 15 actionne le comparateur de mise en forme des données. Selon le réglage du curseur de la résistance de 3 300 Ω, la broche 15 passe à l'état haut lorsque le niveau d'entrée est suffisant, voir la courbe de la **figure 7**. Sur la broche 14 on trouve le signal RSSI que l'on peut utiliser directement pour commander un galvanomètre ou fournir des pips visualisant ainsi le spectre d'une bande de fréquence déterminée par l'action d'un signal de balayage sur l'oscillateur local.

La dent de scie, de largeur et d'amplitude variable est super-

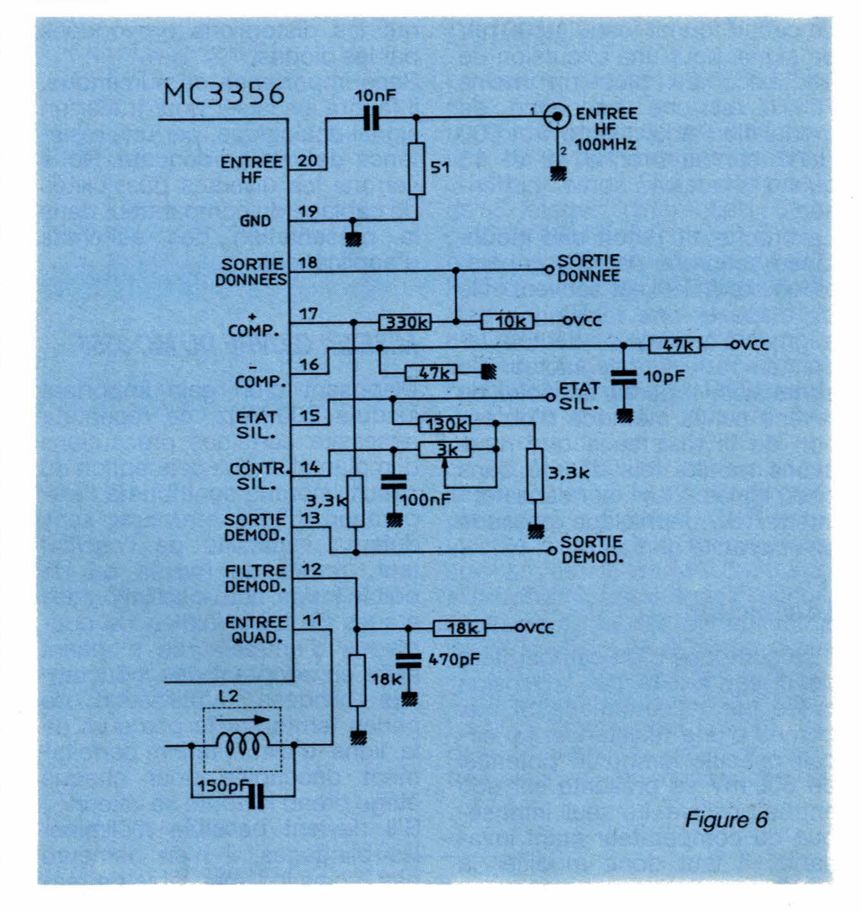


Figure 6

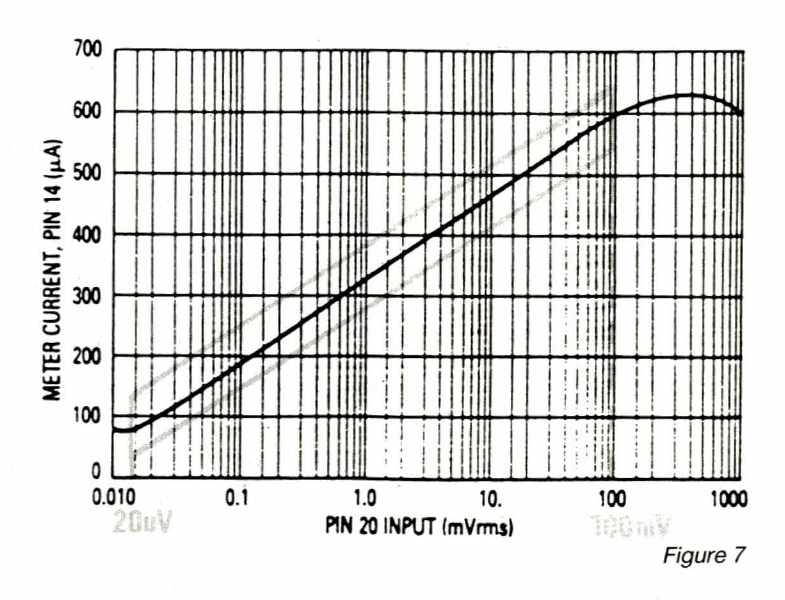


Figure 7

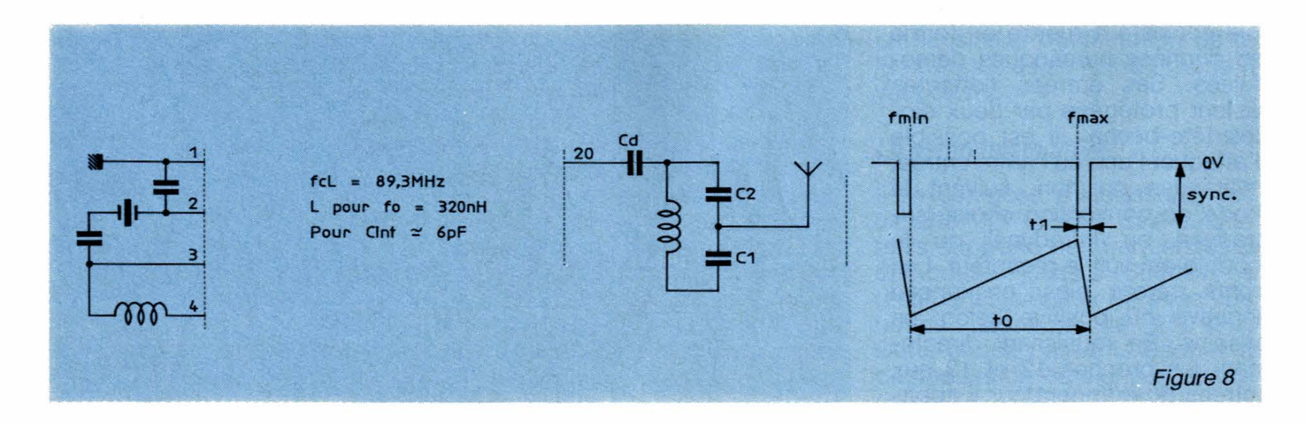
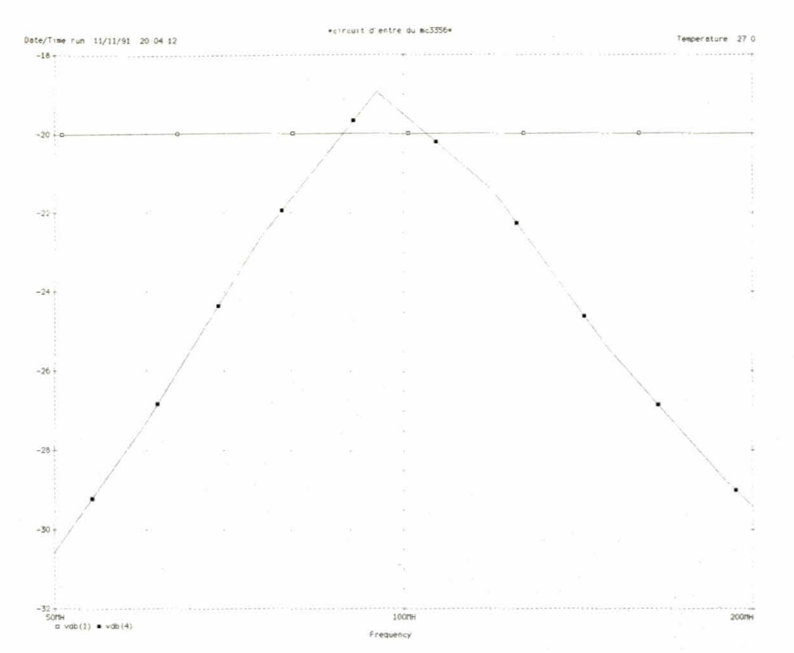


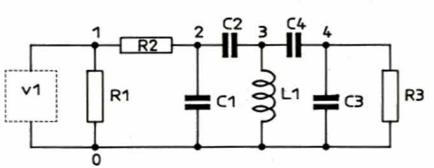
Figure 8



Réponse obtenue avec PSPICE. $C_1 = 46$ pF, $C_2 = 35$ pF.

posée à une tension continue qui fixe la fréquence centrale sur l'écran de l'oscilloscope, on l'applique sur la voie X, le signal RSSI quant à lui va sur la voie Y. On peut aussi créer un signal de synchronisation à amplitude négative superposé au RSSI, afin d'utiliser la double base de temps pour contrôler finement

Circuit d'entrée modélisé sous PSPICE.




```
PSPICE CONTROL SHELL

*circuit d'entrée du mc3356*
.ac dec 10 30meg 300meg
.probe
R1 1 0 10E6
R2 2 1 50
R3 4 0 260
C1 2 0 92E-12
C2 3 2 22E-12
C3 4 0 5E-12
C4 3 4 10E-9
L1 3 0 100E-9
VI 1 0 ac 0.1V
.end
```

une fréquence particulière, voir la **figure 8**.

Le temps t_0 correspond à la vitesse de balayage et t_1 au temps de descente de la dent de scie; à titre indicatif un simple NE 555 permet d'obtenir ces deux signaux aux broches 3 et 7 du circuit.

Le MC 3356 d'apparence complexe à mettre en œuvre n'en reste pas moins très performant et capable de s'adapter à une vaste gamme d'applications. La disponibilité de ce circuit intégré ne saurait être mise en cause d'ici quelques temps.

Ph.-B

